
cookietemple Documentation

Release 1.4.1

Lukas Heumos, Philipp Ehmele

Mar 12, 2022

CONTENTS:

1	cookiетemple overview	3
1.1	Installing	3
1.2	config	3
1.3	list	3
1.4	info	3
1.5	create	4
1.6	lint	4
1.7	bump-version	4
1.8	sync	4
1.9	warp	4
1.10	upgrade	4
1.11	Projects using cookiетemple	5
1.12	Contributing	5
1.13	Authors	5
2	Installation	7
2.1	Stable release	7
2.2	From sources	7
2.3	Upgrading cookiетemple	8
2.4	Windows Installation	8
3	Create a project	9
3.1	Usage	9
3.2	Flags	9
4	Getting information about available templates	11
4.1	list	11
4.1.1	Usage	11
4.2	info	11
4.2.1	Usage	12
5	Linting your project	13
5.1	Usage	14
5.2	Flags	14
5.3	Linting codes	14
5.3.1	General	14
5.3.1.1	general-1	14
5.3.1.2	general-2	14
5.3.1.3	general-3	14
5.3.1.4	general-4	14

5.3.1.5	general-5	15
5.3.1.6	general-6	15
5.3.1.7	general-7	15
5.3.2	cli-python	15
5.3.2.1	cli-python-1	15
5.3.2.2	cli-python-2	16
5.3.2.3	cli-python-3	16
5.3.3	cli-java	16
5.3.3.1	cli-java-1	16
5.3.3.2	cli-java-2	16
5.3.4	lib-cpp	16
5.3.4.1	lib-cpp-1	16
5.3.4.2	lib-cpp-2	16
5.3.5	web-python	17
5.3.5.1	web-python-1	17
5.3.5.2	web-python-2	17
5.3.6	gui-java	17
5.3.6.1	gui-java-1	17
5.3.6.2	gui-java-2	17
5.3.7	pub-thesis	17
5.3.7.1	pub-thesis-1	17
5.3.7.2	pub-thesis-2	17
6	Bumping the version of an existing project	19
6.1	Usage	19
6.2	Flags	20
6.3	Configuration	20
7	Syncing your project	23
7.1	Requirements for sync	23
7.2	Usage	23
7.3	Flags	24
7.4	Configuring sync	24
7.4.1	Enable/Disable sync	24
7.4.2	Sync level	24
7.4.3	Blacklisting files	24
8	Packaging using warp	25
8.1	Warp setup	25
8.2	Usage	26
8.3	Flags	26
9	Configure cookietemple	27
9.1	Usage	27
9.2	Flags	27
9.3	On Github personal access tokens	28
10	Upgrade cookietemple	29
10.1	Usage	29
11	Available templates	31
11.1	cli-python	31
11.1.1	Purpose	31
11.1.2	Design	31
11.1.3	Included frameworks/libraries	33

11.1.4	Usage	33
11.1.5	FAQ	34
11.1.5.1	Do I need a command line interface?	34
11.1.5.2	flake8 and darglint are very slow	34
11.2	cli-java	34
11.2.1	Purpose	34
11.2.2	Design	35
11.2.3	Included frameworks/libraries	36
11.2.4	Usage	37
11.2.5	FAQ	37
11.2.5.1	Can I use cli-java without GraalVM?	37
11.2.5.2	How can I access the build artifacts?	37
11.3	gui-java	38
11.3.1	Purpose	38
11.3.2	Design	38
11.3.3	Included frameworks/libraries	39
11.3.4	Usage	40
11.3.5	FAQ	41
11.4	lib-cpp	41
11.4.1	Purpose	41
11.4.2	Design	41
11.4.3	Included frameworks/libraries	42
11.4.4	Usage	43
11.4.4.1	Installing	43
11.4.4.2	Building the project	43
11.4.4.3	Generating the documentation	43
11.4.4.4	Running tests	44
11.4.5	FAQ	44
11.5	pub-thesis-latex	44
11.5.1	Purpose	44
11.5.2	Design	44
11.5.3	Included frameworks/libraries	46
11.5.4	Usage	46
11.5.4.1	Building your thesis - LaTeX / PDFLaTeX	46
11.5.4.1.1	Using latexmk (Unix/Linux/Windows)	46
11.5.4.1.2	Using the make file (Unix/Linux)	46
11.5.4.1.3	Shell script for PDFLaTeX (Unix/Linux)	47
11.5.4.1.4	Using the batch file on Windows OS (PDFLaTeX)	47
11.5.4.2	Building your thesis - XeLaTeX	47
11.5.4.2.1	Using latexmk (Unix/Linux/Windows)	47
11.5.4.3	Building your thesis - LuaLaTeX	48
11.5.4.3.1	Using latexmk (Unix/Linux/Windows)	48
11.5.4.4	Usage details	48
11.5.4.4.1	Class options	48
11.5.4.4.2	Title page	49
11.5.4.4.3	Abstract separate	49
11.5.4.4.4	Chapter mode	49
11.5.4.4.5	Draft	50
11.5.4.4.6	Choosing the fonts	50
11.5.4.4.7	Choosing the bibliography style	50
11.5.4.4.8	Choosing the page style	51
11.5.4.4.9	Changing the visual style of chapter headings	51
11.5.4.4.10	Custom settings	52
11.5.4.4.11	Nomenclature definition	52

11.5.4.5	Git hooks	53
11.5.4.6	General guidelines	53
11.6	web-website-python	53
11.6.1	Purpose	53
11.6.2	Design	54
11.6.2.1	The basic setup	54
11.6.2.2	The advanced setup	56
11.6.3	Included frameworks/libraries	59
11.6.4	Usage	60
11.6.4.1	The basic template usage	60
11.6.4.2	The advanced template usage	60
11.6.5	Automatic Deployment	61
11.6.6	FAQ	62
11.7	Shared FAQ	62
11.7.1	What are the available domains?	62
11.7.2	How do I publish my documentation?	62
11.7.2.1	Read the Docs	62
11.7.2.2	Github Pages	62
11.7.3	What is Dependabot and how do I set it up?	63
11.7.4	Release Drafter	63
11.7.5	How do I add a new template?	63
12	Github Support	65
12.1	Overview	65
12.2	Branches	65
12.2.1	Overview	65
12.2.2	Branch protection rules	65
12.3	Github Actions	66
12.3.1	Overview	66
12.3.2	main_master_branch_protection workflow	66
12.3.3	release drafter workflow	66
12.3.4	sync_project.yml	66
12.4	Secrets	67
12.4.1	Error Handling due to failed Github repository creation	67
12.5	Issue labels	67
13	Contributing	69
13.1	Types of Contributions	69
13.1.1	Report Bugs	69
13.1.2	Fix Bugs	69
13.1.3	Implement Features	69
13.1.4	Add Templates	70
13.1.5	Write Documentation	70
13.1.6	Submit Feedback	70
13.2	Get Started!	70
13.3	Pull Request Guidelines	71
13.4	Tips	71
14	Adding new templates	73
14.1	Template requirements	73
14.2	Step by step guide to adding new templates	74
15	External Python based projects	81
16	FAQ	83

16.1	I need help with cookietemple. How can I get in contact with the developers?	83
16.2	I am looking for a template for domain x and language y, but it does not exist yet!	83
17	Troubleshooting	85
18	Community	87
18.1	Development Leads	87
18.2	Core contributors	87
18.3	Contributors	87
19	Contributor Covenant Code of Conduct	89
19.1	Our Pledge	89
19.2	Our Standards	89
19.3	Our Responsibilities	89
19.4	Scope	90
19.5	Enforcement	90
19.6	Attribution	90
20	Indices and tables	91



A cookiecutter based project template creation tool supporting several domains and languages with advanced linting, syncing and standardized workflows to get your project kickstarted in no time.

- Documentation: <https://cookietemplate.readthedocs.io> .

COOKIETEMPLATE OVERVIEW

1.1 Installing

Start your journey with cookietemplate by installing it via `$ pip install cookietemplate`.

See [Installation](#).

1.2 config

Configure cookietemplate to get started.

See [Configuring cookietemplate](#)

1.3 list

List all available cookietemplate templates.

See [Listing all templates](#).

1.4 info

Get detailed information on a cookietemplate template.

See [Get detailed template information](#).

1.5 create

Kickstart your customized project with one of cookietemple's templates in no time.

See [Create a project](#).

1.6 lint

Use advanced linting to ensure your project always adheres to cookietemple's standards.

See [Linting your project](#)

1.7 bump-version

Bump your project version with many configurable options.

See [Bumping the version of an existing project](#).

1.8 sync

Sync your project with the latest cookietemple release to get the latest template features.

See [Syncing a project](#).

1.9 warp

Create a self contained executable. Currently, cookietemple does not ship any templates anymore, where this may be required.

See [Warping a project](#).

1.10 upgrade

Check whether you are using the latest cookietemple version and update automatically to benefit from the latest features.

See <https://cookietemple.readthedocs.io/en/latest/upgrade.html>.

1.11 Projects using cookietemplate

- [cookietemplate website](#)
- [system-intelligence](#)
- [mlf-core](#)

1.12 Contributing

cookietemplate is a huge open-source effort and highly welcomes all contributions! Join our [Discord Channel](#). Please read [contributing](#) to find out how you can contribute.

1.13 Authors

cookietemplate was initiated and developed by [Lukas Heumos \(Github\)](#) and [Philipp Ehmele \(Github\)](#). A full list of contributors is available on our [statistics webpage](#).

INSTALLATION

2.1 Stable release

To install cookietemple, run this command in your terminal:

```
$ pip install cookietemple
```

This is the preferred method to install cookietemple, as it will always install the most recent stable release.

If you don't have [pip](#) installed, this [Python installation guide](#) can guide you through the process.

2.2 From sources

The sources for cookietemple can be downloaded from the [Github repo](#).

You can either clone the public repository:

```
$ git clone git://github.com/cookiejar/cookietemple
```

Or download the [tarball](#):

```
$ curl -OJL https://github.com/cookiejar/cookietemple/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ pip install .
```

Alternatively you can also install it using the Makefile:

```
$ make install
```

2.3 Upgrading cookiетemplate

Everytime cookiетemplate is run it will automatically check whether a newer version has been released on PyPI. If a new version has been released you will be informed. To upgrade cookiетemplate either run:

```
$ pip install --upgrade cookiетemplate
```

or by invoking:

```
$ cookiетemplate upgrade
```

For more information please visit [Upgrade cookiетemplate](#).

2.4 Windows Installation

Cookiетemplate exceeds the standard windows path length limit of 260 characters. In order to correctly install cookiетemplate the registry has to be adapted. To open the registry editor run:

```
$ regedit
```

in your command prompt or directly open it through windows search bar.

Consider backing up your current registry state as changes in the registry can always cause problems resulting in your OS not running correctly. You can either export the complete registry by selecting File > Export ... and setting the Export Range flag to 'All' or choose to export only selected branches.

Next find the key '**LongPathsEnabled**' under

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem
```

and change its 'Value Data' from '0' to '1'.

Click 'OK' and close regedit. Continue installing cookiетemplate.

CREATE A PROJECT

Creating projects from templates is the heart of `cookiетemplate`. Our templates adhere to best practices and try to be as modern as possible. Furthermore, they try to automate tasks such as automatical dependency resolvment and installation, packaging, deployment and more. To learn more about our templates please visit [Available templates](#) and check out your template of interest.

3.1 Usage

The creation of a new project can be invoked by

```
$ cookiетemplate create
```

which will guide you through the creation process of your (customized) project via prompts. If you do not have `cookiетemplate` configured yet, you will be asked to do so. For more details please visit [Configure cookiетemplate](#).

The prompts follow the pattern of domain (e.g. `cli`, `gui`, ...), subdomain (if applicable, e.g. `website`), language (e.g. `Python`) followed by template specific prompts (e.g. `testing frameworks`, ...). The project will be created at the current working directory, where `cookiетemplate` has been called.

After the project has been created, linting (see [Linting your project](#)) is automatically performed to verify that the template creation process was successful.

Finally, you will be asked whether you want to automatically push your new project to Github. Note that for this purpose you need to have `cookiетemplate` configured with a Github personal access token. For more details about the Github support please visit [Github Support](#).

3.2 Flags

- `--domain` : To directly create a template of the the corresponding domain.
- `--path [CWD]`: An absolute or relative path to create the template at.

All further prompts will still be asked for. Example: `cli`. It is also possible to directly create a specific template using its handle

GETTING INFORMATION ABOUT AVAILABLE TEMPLATES

Although, information on all cookietemplate templates is provided in [Available templates](#) in our documentation, it is often times more convenient to get a quick overview from the commandline. Hence, cookietemplate provides two commands `list` and `info`, which print information on all available templates with different levels of detail.

4.1 list

`cookietemplate list` is restricted to the short descriptions of the templates. If you want to read more about a specific (sets of) template, please use the `info` command.

All available COOKIETEMPLATE templates				
Name	Handle	Short Description	Available Libraries	Version
Python Commandline Package	cli-python	General Python package with command line interface	click, argparse	1.0.0
Java Commandline Tool	cli-java	A GraalVM based multiplatform commandline application using Picocli.	picocli	1.0.0
Kotlin Commandline Tool	cli-kotlin	someShortDescription	picocli	0.0.1
Python Based Website	web-website-python	someShortDescription	flask	0.0.1
JavaFX GUI with Java	gui-java	someShortDescription	JavaFX	0.0.1
JavaFX GUI with Kotlin	gui-kotlin	someShortDescription	JavaFX	0.0.1
Latex Thesis	pub-thesis-latex	Template for BSc./MSc./PhD/... thesis	latex	1.0.0

Fig. 1: Example output of `cookietemplate list`. Note that the content of the output is of course subject to change.

4.1.1 Usage

`cookietemplate list` can be invoked *via*

```
$ cookietemplate list
```

4.2 info

The `info` command should be used when the short description of a template is not sufficient and a more detailed description of a specific template is required.

Name	Handle	Long Description	Available Libraries	Version
Python Commandline Package	cli-python	Best practice Python package with optional command line support (click, argparse). Pytest or unittest are available as testing libraries. The package can be easily locally installed and distributed on PyPi.	click, argparse	1.0.0

Fig. 2: Example output of `cookietemplate info`.

4.2.1 Usage

Invoke cocketemplate info *via*

```
$ cocketemplate info <HANDLE/LANGUAGE/DOMAIN>
```

- HANDLE : a cocketemplate template handle such as cli-python.
- DOMAIN : a domain for which cocketemplate provides templates for. Example: cli.
- LANGUAGE : A programming language for which cocketemplate provides templates for. Example: python.

LINTING YOUR PROJECT

Linting is the process of statically analyzing code to find code style violations and to detect errors. `cookiecutter` implements a custom linting system, but depending on the template external tools linting tools may additionally be called.

`cookiecutter`'s linting is divided into three distinct phases.

1. All linting functions, which all templates share are called and the results are collected.
2. Template specific linting functions are invoked and the results are appended to the results of phase 1

The linting results of the first two phases are assigned into 3 groups:

1. Passed
2. Passed with warning
3. Failed

If any of the checks failed linting stops and returns an error code.

```
Running general linting
Processing... 100% 0:00:00
Running cli-java linting
Processing... 100% 0:00:00
=====
          LINTING RESULTS
=====
[✓] 5 tests passed
[!] 1 tests had warnings
[x] 0 tests failed
Test passed:
https://cookiecutter/linting/errors#1 : All required general files were found!
https://cookiecutter/linting/errors#1 : File not found check: .travis.yml
https://cookiecutter/linting/errors#2 : Dockerfile check passed
https://cookiecutter/linting/errors#5 : Versions were consistent over all files
https://cookiecutter/linting/errors#1 : All required template specific files were found!
Test Warnings:
https://cookiecutter/linting/errors#3 : TODO string found in usage.rst: Write your usage documentation here.
```

Fig. 1: Linting applied to a newly created cli-java project.

To examine the reason for a failed linting test please follow the URL. All reasons are explained in the section *Linting codes*.

5.1 Usage

cookiетemplate lint can be invoked on an existing project using

```
$ cookiетemplate lint <PATH>
```

- PATH [CWD]: The relative path to the project directory.

5.2 Flags

5.3 Linting codes

The following error numbers correspond to errors found during linting. If you are not sure why a specific linting error has occurred you may find more information using the respective error code.

5.3.1 General

5.3.1.1 general-1

File not found. This error occurs when your project does not include all of cookiетemplate's files, which all templates share.

Please create the file and populate it with appropriate values. You should also critically reflect why it is missing, since at the time of the project creation using cookiетemplate this file should not have been missing!

5.3.1.2 general-2

Dockerfile invalid. This error usually originates from empty Dockerfiles or missing FROM statements.

5.3.1.3 general-3

TODO string found. The origin of this error are COOKIETEMPLATE TODO: or TODO COOKIETEMPLATE: strings in the respective files. Usually, they point to things that should be manually configured or require other attention. You may remove them if there is no task for you to be solved.

5.3.1.4 general-4

Cookiecutter string found. This error occurs if something went wrong at the project creation stage. After a project has been created using cookiетemplate there should not be any jinja2 syntax statements left. Web development templates may pose exceptions. However, {{ *cookiecutter* }} statements should definitely not be present anymore.

5.3.1.5 general-5

Versions not consistent. If the version of all files specified in the [bumpversion] sections defined in the qube.cfg file are not consistent, this error may be found. Please ensure that the version is consistent! If you need to exclude specific lines from this check please consult *Bumping the version of an existing project*. To prevent this error you should only increase the version of your project using `cookiетemplate bump-version`.

5.3.1.6 general-6

changelog.rst invalid. The changelog.rst file requires that every changelog section has a header with the version and the corresponding release date. The version above another changelog section should always be *greater* than the section below (e.g. 1.1.0 above 1.0.0). Every section must have the headings ****Added****, ****Fixed****, ****Dependencies**** and ****Deprecated****.

5.3.1.7 general-7

cookiетemplate.cfg linting failed. The cookiетemplate.cfg plays a major role in cookiетemplate’s sync and bump-version functionality.

The linter ensures that following requirements are met:

- 1.) Every config file should have at least the following sections: bumpversion, bumpversion_files_whitelisted, bumpversion_files_blacklisted, sync_files_blacklisted, sync_level
- 2.) bumpversion should only contain a current_version value (the project’s current version)
- 3.) bumpversion_files_whitelisted should contain at least the .cookiетemplate.yml file in the dot_cookiетemplate variable
- 4.) sync_level should only contain a ct_sync_level value (and this value should be one of either patch, minor or major)
- 5.) sync_files_blacklisted should contain at least the CHANGELOG.rst file (excluding it from syncing to avoid PR updates)
- 6.) sync should only contain a sync_enabled value (and this value should be one of either True|true|Yes|yes|Y|y|False|false|No|no|N|n)

5.3.2 cli-python

5.3.2.1 cli-python-1

File not found. This error occurs when your project does not include all of cli-python’s expected files.

Please create the file and populate it with appropriate values. You should also critically reflect why it is missing, since at the time of the project creation using cookiетemplate this file should not have been missing!

5.3.2.2 cli-python-2

PyPI dependency not up to date. The dependencies specified in the requirements.txt and requirements_dev.txt are not up to date.

It is up to you whether you can and want to update them.

5.3.2.3 cli-python-3

The cookiетemple.cfg section called sync_files_blacklisted misses either requirements = requirements.txt, requirements_dev = requirements_dev.txt or changelog = CHANGELOG.rst. All are required to exclude them from syncing and interference with services like dependabot.

5.3.3 cli-java

5.3.3.1 cli-java-1

File not found. This error occurs when your project does not include all of cli-java's expected files.

Please create the file and populate it with appropriate values. You should also critically reflect why it is missing, since at the time of the project creation using cookiетemple this file should not have been missing!

5.3.3.2 cli-java-2

The cookiетemple.cfg section called sync_files_blacklisted misses build_gradle = gradle.build or changelog = CHANGELOG.rst. Both are required to exclude the gradle build file from syncing.

5.3.4 lib-cpp

5.3.4.1 lib-cpp-1

File not found. This error occurs when your project does not include all of lib-cpp's expected files.

Please create the file and populate it with appropriate values. You should also critically reflect why it is missing, since at the time of the project creation using cookiетemple this file should not have been missing!

5.3.4.2 lib-cpp-2

The cookiетemple.cfg section called sync_files_blacklisted misses changelog = CHANGELOG.rst. This is required to be excluded from syncing.

5.3.5 web-python

5.3.5.1 web-python-1

File not found. This error occurs when your project does not include all of web-python’s expected files. Please create the file and populate it with appropriate values. You should also critically reflect why it is missing, since at the time of the project creation using cookiетemplate this file should not have been missing!

5.3.5.2 web-python-2

The cookiетemplate.cfg section called `sync_files_blacklisted` misses either `requirements = requirements.txt`, `requirements_dev = requirements_dev.txt` or `changelog = CHANGELOG.rst`. All are required to exclude them from syncing and interference with services like dependabot.

5.3.6 gui-java

5.3.6.1 gui-java-1

File not found. This error occurs when your project does not include all of gui-java’s expected files. Please create the file and populate it with appropriate values. You should also critically reflect why it is missing, since at the time of the project creation using cookiетemplate this file should not have been missing!

5.3.6.2 gui-java-2

The cookiетemplate.cfg section called `sync_files_blacklisted` misses `pom = pom.xml` or `changelog = CHANGELOG.rst`. Both are required to be excluded from syncing.

5.3.7 pub-thesis

5.3.7.1 pub-thesis-1

File not found. This error occurs when your project does not include all of pub-thesis’s expected files. Please create the file and populate it with appropriate values. You should also critically reflect why it is missing, since at the time of the project creation using cookiетemplate this file should not have been missing!

5.3.7.2 pub-thesis-2

The cookiетemplate.cfg section called `sync_files_blacklisted` misses `changelog = CHANGELOG.rst`. This is required to be excluded from syncing.

BUMPING THE VERSION OF AN EXISTING PROJECT

Increasing the version of an already existing project is often times a cumbersome and error prone process, since the version has to be changed in multiple places. To facilitate this process, cookiecutter provides a `bump-version` command, which conveniently increases the version across several files and commits them. Additionally, `bump-version` inserts a new section into the changelog using the specified new version.

```
New Version (Date)
```

```
-----
```

```
**Added**
```

```
**Fixed**
```

```
**Dependencies**
```

```
**Deprecated**
```

`bump-version` will verify that your new version adheres to [semantic versioning](#) and that you are not trying to update it unreasonably. It is for example not allowed to bump from 2.0.0 to 7.1.2, since in a normal development workflow only 2.0.1, 2.1.0 or 3.0.0 adhere to consecutive [semantic versioning](#). Note that SNAPSHOT versions are allowed! However, it must still follow [semantic versioning](#). Version 1.2.5 therefore cannot be the predecessor of 1.2.5-SNAPSHOT, but only 1.2.4.

6.1 Usage

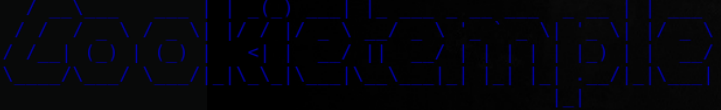
The `bump-version` command follows the syntax

```
$ cookiecutter bump-version <OPTIONS> X.X.X <PATH>
```

- **X.X.X**: The new version, where the X correspond to integers adhering to consecutive [semantic versioning](#). You may append `-SNAPSHOT`.
- **PATH [CWD]**: The path to the `cookiecutter.cfg` file, which contains all locations, where the version should be increased.

Note that you can use `bump-version` without passing any parameters. This way, cookiecutter will let you choose from three valid options to bump your projects version. Note that this will only work in the main directory of your project due to some cli constraints.

```
zeth@master /tmp> cookietemple bump-version 1.0.0 Exploding_Springfield
```



```
Run cookietemple --help for an overview of all commands

Changing version number.
Current version is 0.1.0.
New version will be 1.0.0

Updating version number in Exploding_Springfield/setup.py
- version='0.1.0',
+ version='1.0.0',

Updating version number in Exploding_Springfield/Exploding_Springfield/__init__.py
- __version__ = '0.1.0'
+ __version__ = '1.0.0'

Updating version number in Exploding_Springfield/.cookietemple.yml
- version: 0.1.0
+ version: 1.0.0

Staging template.
Committing changes to local git repository.
```

Fig. 1: bump-version applied to a fresh cli-python project

6.2 Flags

- `--downgrade` : To downgrade a version.

The changelog won't be modified. Only use this option as a last resort if something went horribly wrong in your development process. In a normal development workflow, this should never be necessary.

- `--project-version`: To get the current project version.

No version bumping will be triggered. Using this flag will cancel any commands executed after and exits the program.

- `--tag` or `-t` : To tag the bump version commit.

One can use this flag to tag the current bump version commit with the updated version for reuse in releases. Note that this will require to be pushed from local to remote by using `git push origin <tagname>`.

6.3 Configuration

All templates of `cookietemplate` ship with a `cookietemplate.cfg` file, which defines all files `bump-version` examines.

The bump-version configuration begins with the section:

```
[bumpversion]
current_version = 0.1.0
```

where the current version is defined. All files are either white- or blacklisted (see below for explanations). An arbitrary name is followed by the path to the file: `arbitrary_name = path_to_file`.

Whitelisted files are listed below a [bumpversion_files_whitelisted] section, e.g.:

```
[bumpversion_files_whitelisted]
dot_cookietemple = .cookietemple.yml
conf_py = docs/conf.py
```

All files, which are whitelisted are searched for patterns matching `X.X.X`, which are updated to the specified new versions. Any lines, which contain the string `<<COOKIETEMPLATE_NO_BUMP>>` will be ignored.

If files, like Maven `pom.xml` files, contain many version patterns matching `X.X.X`, it may be a better idea to blacklist them (section `[bumpversion_files_blacklisted]`) and enable only specific lines to be updated:

```
[bumpversion_files_blacklisted]
pom = pom.xml
```

Analogously to whitelisted files, which allow for specific lines to be ignored, blacklisted files allow for specific lines to be forcibly updated using the string `<<COOKIETEMPLATE_FORCE_BUMP>>`.

Note that those tags must be on the same line as the version (commonly placed in a comment), otherwise they won't work!

SYNCING YOUR PROJECT

Syncing is supposed to integrate any changes to the cookietemple templates back into your already existing project. When `cookietemple sync` is invoked, cookietemple checks whether a new version of the corresponding template for the current project is available. If so, cookietemple creates a temporary project with the most recent template and pushes it to the `TEMPLATE` branch. Next, a pull request is submitted to the `development` branch. Please note that the required `CT_SYNC_TOKEN` (see below) is automatically set and manual syncing should be avoided if possible.

The syncing process is configurable by setting the desired lower syncing boundary level and blacklisting files from syncing (see [Enable/Disable sync](#)).

7.1 Requirements for sync

For syncing to work properly, your project has to satisfy a few things:

1. A Github repository with your projects code (private or public, organization or non-organization repository).
2. An unmodified `.cookietemple.yml` file. If you modify this file, which you should never do, syncing may not be able to recreate the project with the most recent template.
3. An active repository secret called `CT_SYNC_TOKEN` for your project's repository containing the encrypted personal access token with at least `repo` scope.
4. A running, unmodified workflow called `sync_project.yml`. Modifying this workflow should never be done and results in undefined sync behaviour.

Points 3 and 4 are only required when not syncing manually.

7.2 Usage

To sync your project manually, simply run

```
$ cookietemple sync [PROJECT_DIR] [PAT] [GITHUB_USERNAME]
```

- `PROJECT_DIR [CWD]` : The path to the `cookietemple.cfg` file.
- `PAT [Configured pat]` : A Github personal access token with at least the `repo` scope. The `sync_project.yml` Github workflow uses the `PAT` set as a Github secret.
- `GITHUB_USERNAME [Configured username]` : The Github username to submit a pull request from. The supplied `PAT` has to be associated with this username.

7.3 Flags

- `--set-token` : Update `CT_SYNC_SECRET` of your project's repo to a new PAT. Note that the Github username and the PAT must still match for automatic syncing to work.
- `check-update` : Check, whether a new release of a template for an already existing project is available.

7.4 Configuring sync

7.4.1 Enable/Disable sync

Cookiетemplate aims to provide the user as much configuration as possible. So, the sync feature is optional and should also be switched on or off. If you want to enable sync (which is the default), the `sync_enable` accepts the following values: `True`, `true`, `Yes`, `yes`, `Y`, `y`. To disable sync, simply change this value into one of `False`, `false`, `No`, `no`, `N`, `n`. It can be configured in the:

```
[sync]
sync_enable = True
```

section.

7.4.2 Sync level

Since cookiетemplate strongly adheres to semantic versioning our templates do too. Hence, it is customizable whether only major, minor or patch releases of the template should trigger cookiетemplate sync. The sync level therefore specifies a lower boundary. It can be configured in the:

```
[sync_level]
ct_sync_level = minor
```

section.

7.4.3 Blacklisting files

Although, cookiетemplate only submits pull requests for files, which are part of the template, sometimes even those files should be ignored. Examples could be any html files, which, at some point, contain only custom content and should not be synced. When syncing, cookiетemplate examines the `cookiетemplate.cfg` file and ignores any file patterns (globs) (e.g. `*.html`) below the `[sync_files_blacklisted]` section. IMPORTANT NOTE: If you would like to add some files to this section, make sure your current branch (if you are syncing manually, which is not recommended) or your default branch has the latest blacklisted sync file section with your changes, so it will be used by the sync.

PACKAGING USING WARP

cookiетemple ships with Rust binaries of **Warp** for the three major operating systems, Linux, MacOS and Windows. Warp can be called when complex output scripts with dependencies should be merged into single, distributable binaries. An example would be the output of **jlink** applied to modular Java projects. However, **warp** can also be applied to .NET Core projects, NodeJS and others, making it more flexible than e.g. the with Java 14 introduced JPackager.

```
zeth@master /tmp> cookietemple warp --input_dir hellofx/target/hellofx/ --exec bin/launcher --output java_gui.bin

  Cookiетemple

Run cookietemple --help for an overview of all commands

Packaging using warp-packer version: 0.3.0
For more details please visit:  https://github.com/dgiagio/warp    for more information.
Detected linux
/home/zeth/anaconda3/envs/cookiетemple/lib/python3.8/site-packages/cookiетemple-0.1.0-py3.8.egg/cookiетemple/package_dist/warp/linux-x64.warp-packer
is already executable! Will not attempt to change permissions.
Compressing input directory "hellofx/target/hellofx/"...
Creating self-contained application binary "java_gui.bin"...
All done
```

Fig. 1: Example output of **cookiетemple warp** applied to a (former) gui-java project. The project was first packaged using **mvn javafx:jlink** and then warp was applied. Please note the relative path for **--exec**. Note that this is not necessary for GraalVM based projects.

The resulting binary is self contained and does not have any additional dependencies. Note however, that the binaries are **not** cross platform. You need to compile and package on the target platform. For more information please read the **Warp README**. Currently no cookietemple template requires Warp.

8.1 Warp setup

Warp for all major platforms (Linux, Windows 10+, MacOS) is already shipped with cookietemple. Hence, there is no need to install the Warp externally. However, the first time that you invoke Warp you may be asked for your sudo/administrator password, since the Warp executable needs to be granted executable rights. You should only be prompted once, since this setting is permanent. If you update cookietemple or reinstall, the Warp executable may be replaced and you once again need to provide it the required rights.

8.2 Usage

Invoke warp by running

```
$ cookietemple warp --input_dir <INPUTDIR> --exec <EXECUTABLE> --output <OUTPUT>
```

8.3 Flags

- `input-dir`: The path to the directory to package.
- `--exec`: A relative path from the packaged folder to the executable. Please note that the `--exec` operates relative to the packaged folder and may result in ‘file not found’ errors, if a wrongly relative path is given!
- `--output`: A path to the output directory.

CONFIGURE COOKIETEMPLATE

To prevent frequent prompts for information, which rarely or never changes at all such as the full name, email or Github username of the user, cookietemplate uses a configuration file. Moreover, the personal access token associated with the Github username is stored, in encrypted form, to allow for various Github functionalities, such as automatic Github repository creation or *Syncing your project*. The creation of projects with cookietemplate requires a configuration file. A personal access token is not required, if Github support is not used. The configuration file is saved operating system dependent in common config file locations (~/.config/cookietemplate on Unix, C:\Users\Username\AppData\Local\cookietemplate\cookietemplate on Windows). Configuring cookietemplate is only required once, although it is always possible to update the current configuration.

9.1 Usage

Invoke cookietemplate config *via*

```
$ cookietemplate config <all/general/pat>
```

- **all** : Prompt for the full name, email, Github username, whether to create a cookietemplate topic for repos or not and Github personal access token.
- **general** : Only prompt for the full name, email, the Github username and whether to create a cookietemplate topic in repos or not.

These details are required to create projects.

- **pat** : Solely prompts for the Github personal access token and updates it if already set.

Ensure that your Github username still matches with the new personal access token. If not you should also update your Github username *via* `cookietemplate config general`. Additionally, any of your already created projects may still feature your old token and you may therefore run into issues when attempting to push. Hence, you must also [update your remote URL](#) for those projects!

9.2 Flags

- **--view** : To get your current cookietemplate configuration.

The explicit value of your Github personal access token will not be printed. You will only be informed about whether it is set or not.

9.3 On Github personal access tokens

cookietemple's Github support requires access to your Github repositories to create repositories, add issues labels and set branch protection rules. Github manages these access rights through Personal Access Tokens (PAT). If you are using cookietemple's Github support for the first time `cookietemple config pat` will be run and you will be prompted for your Github PAT. Please refer to the [official documentation](#) on how to create one. cookietemple requires `repo` access and `workflow`. This ensures that your PAT would not even allow for the deletion of repositories. cookietemple then encrypts the Personal Access Token, adds the encrypted token to the `cookietemple_conf.cfg` file and saves the key locally in a hidden place. This is safer than Github's official way, which recommends the usage of environment variables or Github Credentials, which both save the token in plaintext. It is still strongly advised to secure your personal computer and not allow any foe to get access.

UPGRADE COOKIETEMPLATE

Every time cookietemplate is run it will automatically contact PyPI to check whether the locally installed version of cookietemplate is the latest version available. If a new version is available cookietemplate can be trivially upgraded. Note that `pip` must be available in your `PATH`. It is advised not to mix installations using `setuptools` directly and `pip`. If you are not a developer of cookietemplate this should not concern you.

10.1 Usage

```
$ cookietemplate upgrade
```


AVAILABLE TEMPLATES

cookiecutter currently has the following templates available:

1. *cli-java*
2. *cli-python*
3. *gui-java*
4. *lib-cpp*
5. *pub-thesis-latex*
6. *web-website-python*

In the following every template is devoted its own section, which explains its purpose, design, included frameworks/libraries, usage and frequently asked questions. A set of frequently questions, which all templates share see here: [Shared FAQ](#) FAQ. It is recommended to use the sidebar to navigate this documentation, since it is very long and cumbersome to scroll through.

11.1 cli-python

11.1.1 Purpose

cli-python is a [Python](#) based template designed for command line applications, but it may also be easily used as standard Python package without any command line interface. It is an improved version of [cookiecutter-hypermodern-python](#).

11.1.2 Design

The Python package is based on a standard [poetry](#) structure with a corresponding `pyproject.toml` and `poetry.lock` file.

```
— AUTHORS.rst
— .bandit.yml
— codecov.yml
— CODE_OF_CONDUCT.rst
— cookiecutter.cfg
— .cookiecutter.yml
— .darglint
— Dockerfile
— docs
```

(continues on next page)

(continued from previous page)

```

├── authors.rst
├── code_of_conduct.rst
├── conf.py
├── index.rst
├── installation.rst
├── make.bat
├── Makefile
├── readme.rst
├── reference.rst
├── requirements.txt
├── _static
│   └── custom_cookietemplate.css
├── usage.rst
├── .editorconfig
├── .flake8
├── .gitattributes
├── .github
│   ├── dependabot.yml
│   ├── ISSUE_TEMPLATE
│   │   ├── bug_report.md
│   │   ├── feature_request.md
│   │   └── general_question.md
│   ├── labels.yml
│   ├── pull_request_template.md
│   ├── release-drafter.yml
│   └── workflows
│       ├── build_package.yml
│       ├── check_no_SNAPSHOT_master.yml
│       ├── check_patch_release_master_only.yml
│       ├── constraints.txt
│       ├── labeler.yml
│       ├── publish_docs.yml
│       ├── publish_package.yml
│       ├── run_cookietemplate_lint.yml
│       ├── run_tests.yml
│       ├── release-drafter.yml
│       └── sync_project.yml
├── .gitignore
├── LICENSE
├── Makefile
├── makefiles
│   ├── Linux.mk
│   └── Windows.mk
├── mypy.ini
├── noxfile.py
├── poetry.lock
├── .pre-commit-config.yaml
├── .prettierignore
├── pyproject.toml
├── README.rst
├── .readthedocs.yml
└── project_name

```

(continues on next page)


```
<<your_project_name>>
```

Run all pre-commit tests with:

```
make test-all
```

Ensure that you have `nox nox-poetry` installed (as specified in the `.github/workflows/constraints.txt` file. Other make targets include:

```
make clean
```

which removes all build files:

```
make build
```

which builds source and wheel packages, which can then be used for a PyPi release using:

```
make release
```

All possible Makefile commands can be viewed using:

```
make help
```

11.1.5 FAQ

11.1.5.1 Do I need a command line interface?

No you do not need a command line interface. `cli-python` can also be used as a Python package. Simply remove all command line related code. At some point we will try to offer a version without a command line interface.

11.1.5.2 flake8 and darglint are very slow

This is a known issue with Google and Numpy doc styles: <https://github.com/terrencepreilly/darglint/issues/186> If this is a concern to you feel free to remove darglint.

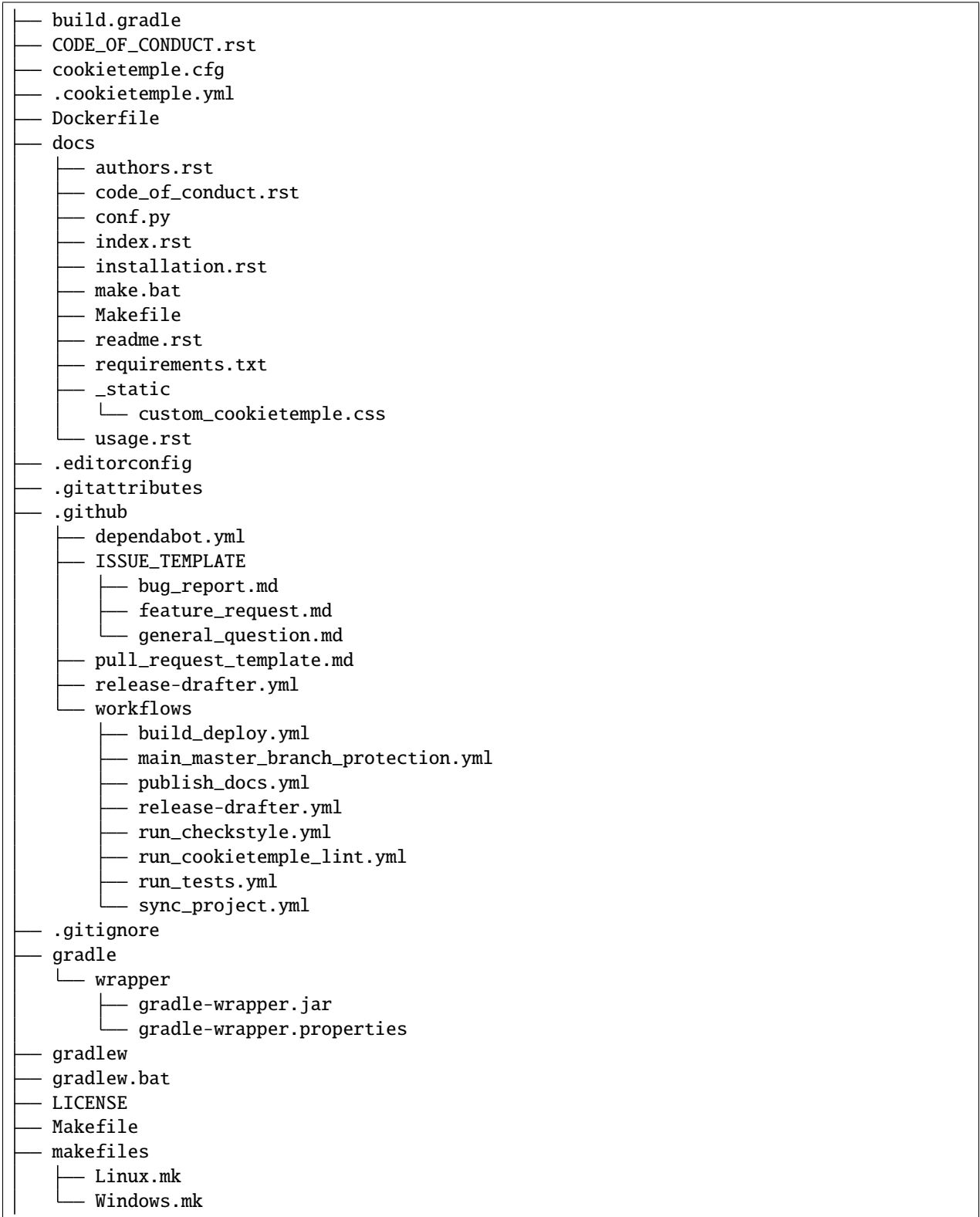
11.2 cli-java

11.2.1 Purpose

`cli-java` is a `Java` based template for designed for command line applications, which require a fast startup time. Hence, it is based on `GraalVM`, which allows for the packaging of the application into small, native self-contained executables. `Picocli` is used as main library for the design of the command line interface.

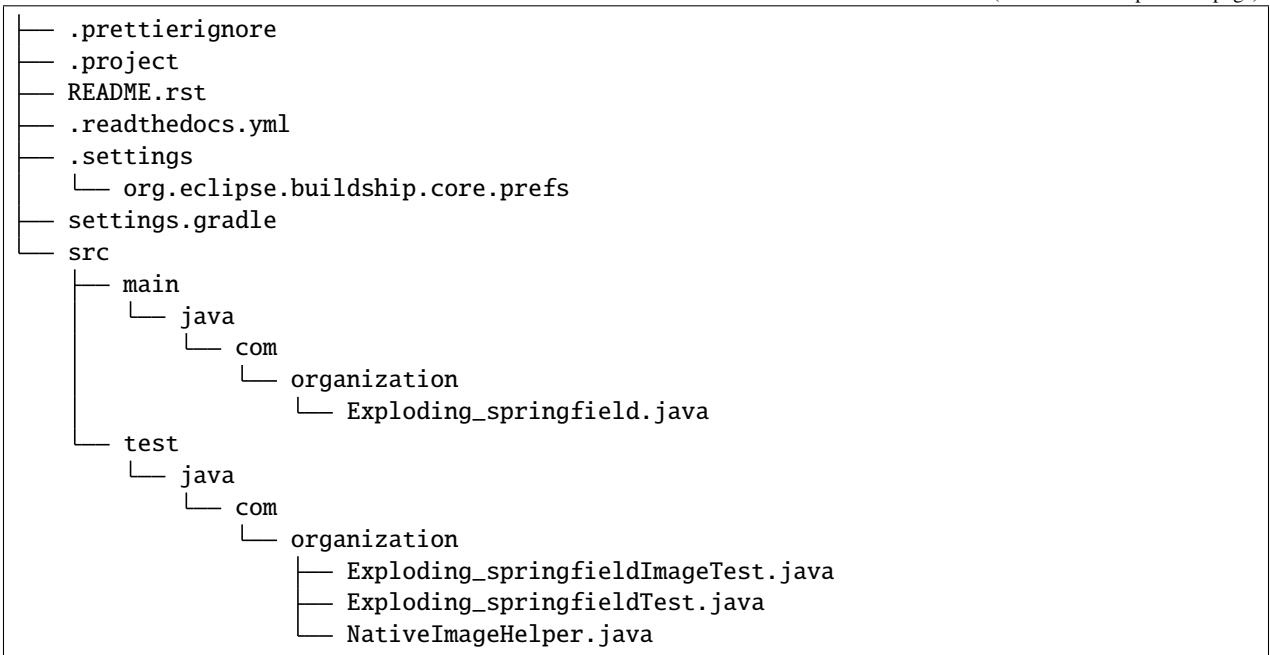
11.2.2 Design

The template is based on a standard [Maven directory structure](#). However, it is using [Gradle](#) as build tool.



(continues on next page)

(continued from previous page)



11.2.3 Included frameworks/libraries

1. [Gradle](#) as build tool
2. [GaalVM](#) as main JDK and virtual layer to allow for native binaries
3. [GaalVM Native Image](#) to build platform dependent self-contained executables
4. [JUnit 5](#) as main testing framework
5. [Picocli](#) to implement the command line interface
6. Preconfigured [readthedocs](#)
7. Seven Github workflows:
 1. `build_docs.yml`, which builds the Read the Docs documentation.
 2. `build_deploy.yml`, which builds the cli-java project into Linux, MacOS and Windows executables. They are deployed as build artifacts.
 3. `run_checkstyle.yml`, which runs [checkstyle](#) linting using Google's ruleset.
 4. `run_tests.yml`, which runs all JUnit tests.
 5. `main_master_branch_protection`: Please read [main_master_branch_protection workflow](#).
 6. `release-drafter.yml`: Please read [release drafter workflow](#).
 7. `run_cookietemple_lint.yml`, which runs `cookietemple lint` on the project.
 8. `sync_project.yml`, which syncs the project to the most recent cookietemple template version

11.2.4 Usage

cli-java requires you to have [Gradle](#), [GraalVM](#) and [GraalVM Native Image](#) installed. Please follow the instructions on the respective websites to install them. Ensure that GraalVM is the default JDK by running `java -version`

A platform dependent executable (of the current running operating system!) can then be build by invoking:

```
make binary
```

or alternatively:

```
gradle build
```

Your platform dependent executable can then be found in the folder `build/native-image`.

Alternatively you can directly build and run your binary by invoking:

```
make run
```

All tests can be run by:

```
make test
```

Other make targets include:

```
make clean
```

which removes all build files:

```
make dist
```

All possible Makefile commands can be viewed using:

```
make help
```

11.2.5 FAQ

11.2.5.1 Can I use cli-java without GraalVM?

cli-java is purposefully designed with GraalVM and native images in mind. We advise against using it without GraalVM.

11.2.5.2 How can I access the build artifacts?

Go to the Github Actions tab, select the `build_deploy` workflow and there you can find the artifacts. Note that the workflow must have completed successfully for all operating systems.

11.3 gui-java

11.3.1 Purpose

gui-java is a modular [JavaFX](#) based template to build cross platform Desktop graphical user interfaces (GUIs). It uses [Apache Maven](#) to compile the package and *Packaging using warp* to distribute binaries containing a Java Runtime Environment (JRE).

11.3.2 Design

The template follows the [standard Maven directory layout](#). Therefore, all dependencies are defined in the `pom.xml` file, the tool source code is in `src/java` and the tests in `src/test`.

Please be aware that gui-java is a modular Java 11+ project, which requires a few modifications to distribute and build JavaFX applications. As a result, binaries are a lot smaller. Assuming that your organization is called `cookiejardealer`, the file tree looks as follows:

```

— CODE_OF_CONDUCT.rst
— cookietemplate.cfg
— .cookietemplate.yml
— Dockerfile
— docs
  — authors.rst
  — code_of_conduct.rst
  — conf.py
  — index.rst
  — installation.rst
  — make.bat
  — Makefile
  — readme.rst
  — requirements.txt
  — _static
    — custom_cookietemplate.css
  — usage.rst
— .editorconfig
— .github
  — dependabot.yml
  — ISSUE_TEMPLATE
    — bug_report.md
    — feature_request.md
    — general_question.md
  — pull_request_template.md
  — release-drafter.yml
  — workflows
    — compile_package.yml
    — main_master_branch_protection.yml
    — publish_docs.yml
    — release-drafter.yml
    — run_cookietemplate_lint.yml
    — run_java_linting.yml

```

(continues on next page)

(continued from previous page)



11.3.3 Included frameworks/libraries

1. [Apache Maven](#) to build and solve dependencies
2. [JavaFX \(14\)](#) to build a graphical user interface
3. [JavaFX Maven plugin](#) to build a modular package with a JRE
4. [Packaging using warp](#) to create a single, distributable, platform specific binary
5. [JUnit 5](#) for unit tests
6. [TestFX](#) for JavaFX GUI tests
7. Preconfigured [readthedocs](#)
8. Eight Github workflows:
 1. `build_docs.yml`, which builds the readthedocs documentation.
 2. `compile_package.yml`, which compiles the gui-java project.
 3. `run_java_linting.yml`, which runs [checkstyle](#) linting using Google's ruleset.

4. `run_tests.yml`, which runs the Unit tests. Note that this workflow is currently disabled, since GUI unittests are not possible using Github Actions.
5. `run_codecov`, apply codecov to your project/PRs in your project and create automatically a report with the details at codecov.io
6. `main_master_branch_protection`: Please read [main_master_branch_protection workflow](#).
7. `release-drafter.yml`: Please read [release drafter workflow](#).
8. `run_cookietemple_lint.yml`, which runs `cookietemple lint` on the project.
8. `sync_project.yml`, which syncs the project to the most recent cookietemple template version.

11.3.4 Usage

The usage of `gui-java` is primarily Makefile based. Please be aware that you need [Apache Maven](#) and Java 11+ installed.

All (Maven) commands are wrapped into Make commands, but can of course also be called directly:

The generated `gui-java` project can be installed using:

```
make install
```

Other make targets include:

```
make clean
```

which removes all build files:

```
make dist
```

which runs `jlink` to build the `gui-java` project with a custom platform dependent JRE. Be aware, that this results in six folders. The executable binary can be found in the `target/bin` folder and is called `launcher`.

If you want to package the resulting custom JRE together with the launcher and all other required files (aka the six folders), then run the:

```
make binary
```

`goal. make binary` calls the `make dist` goal and then packages the files into a single, platform dependent executable using [Packaging using warp](#). This executable can then be easily distributed.

Tests can be run via:

```
make test
```

All possible Makefile commands can be viewed using:

```
make help
```


11.3.5 FAQ

None yet.

11.4 lib-cpp

11.4.1 Purpose

A template for modern C++ projects - both executables and libraries - using CMake, Clang-Format, CI, unit testing and more, with support for downstream inclusion.

11.4.2 Design

The template is inspired by several others (mainly [TheLartians'](#) and [Jason Turner's](#) [<https://github.com/lefticus/cpp_starter_project>](https://github.com/lefticus/cpp_starter_project)). It is using [CMake](#) as its build system.

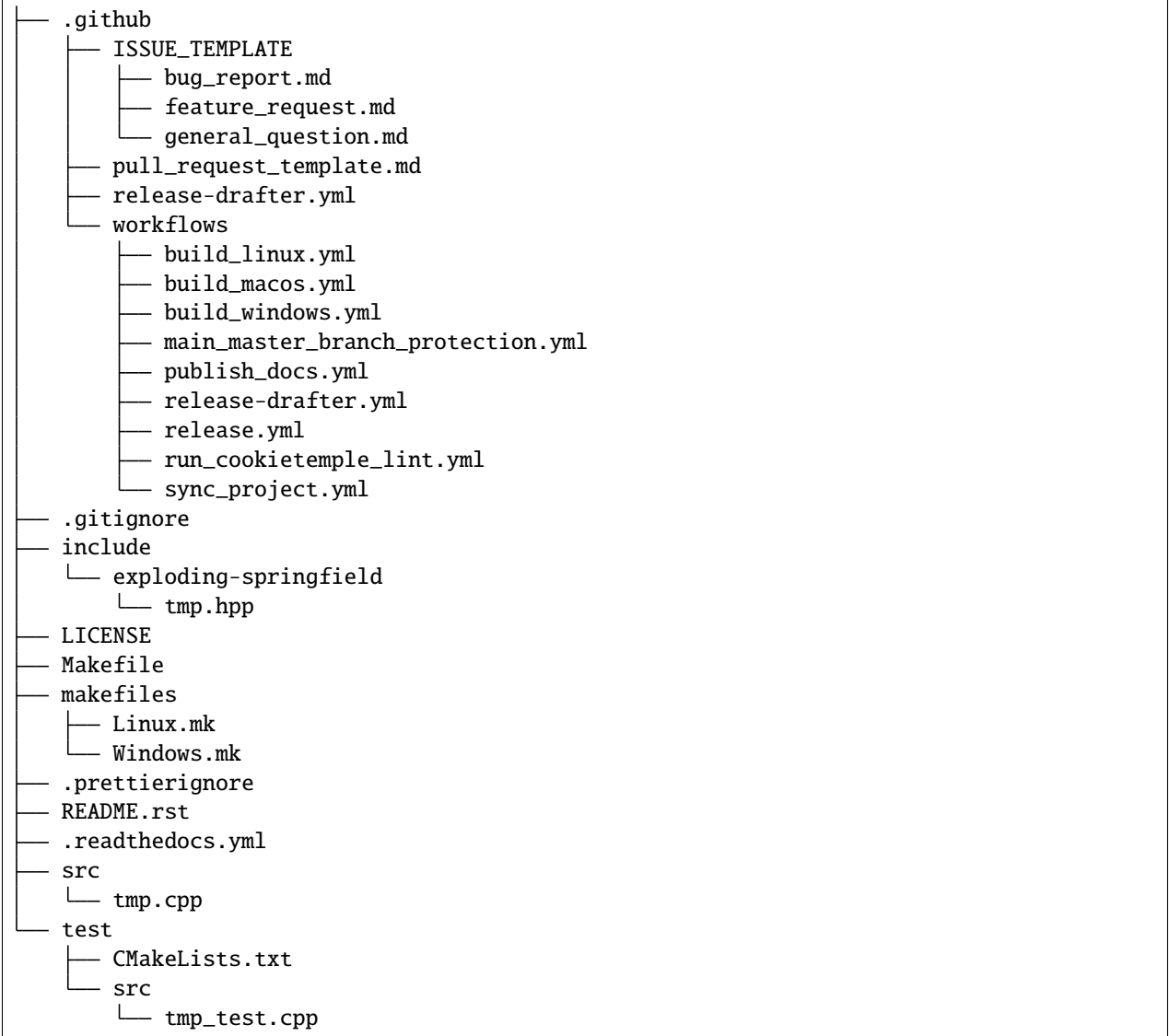
```

— .clang-format
— .clang-tidy
— cmake
  — CompilerWarnings.cmake
  — Conan.cmake
  — Doxygen.cmake
  — exploding-springfieldConfig.cmake.in
  — SourcesAndHeaders.cmake
  — StandardSettings.cmake
  — StaticAnalyzers.cmake
  — Utils.cmake
  — Vcpkg.cmake
  — version.hpp.in
— CMakeLists.txt
— codecov.yaml
— CODE_OF_CONDUCT.rst
— CONTRIBUTING.rst
— cookietemple.cfg
— .cookietemple.yml
— Dockerfile
— docs
  — authors.rst
  — code_of_conduct.rst
  — conf.py
  — index.rst
  — installation.rst
  — make.bat
  — Makefile
  — readme.rst
  — requirements.txt
  — _static
    — custom_cookietemple.css
  — usage.rst
— .editorconfig

```

(continues on next page)

(continued from previous page)



11.4.3 Included frameworks/libraries

1. Modern **CMake** configuration and project
2. An example of a **Clang-Format** config, inspired from the base *Google* model, with minor tweaks.
3. **Static analyzers** integration, with *Clang-Tidy* and *Cppcheck*, the former being the default option
4. **Doxygen** support, through the `ENABLE_DOXYGEN` option, which can enable if desired_config
5. **Unit testing** support, through *GoogleTest* (with an option to enable *GoogleMock*) or *Catch2*
6. **Code coverage**, enabled by using the `ENABLE_CODE_COVERAGE` option, through *Codecov* CI integration
7. **Package manager support**, with *Conan* and *Vcpkg*, through their respective options
8. **CI workflows for Windows, Linux and MacOS** using *GitHub Actions*, making use of the caching features, to ensure minimum run time

9. Options to build as a header-only library or executable, not just a static or shared library
10. CCache integration, for speeding up build times

11.4.4 Usage

11.4.4.1 Installing

To install an already built project, you need to run the `install` target with CMake. For example:

```
cmake --build build --target install --config Release

# a more general syntax for that command is:
cmake --build <build_directory> --target install --config <desired_config>
```

11.4.4.2 Building the project

To build the project, all you need to do, **after correctly `installing the project <README.rst#Installing>`**, is run a similar CMake routine to the the one below:

```
mkdir build/ && cd build/
cmake .. -DCMAKE_INSTALL_PREFIX=/absolute/path/to/custom/install/directory
cmake --build . --target install
```

Note: The custom `CMAKE_INSTALL_PREFIX` can be omitted if you wish to install in the default install location.

More options that you can set for the project can be found in the `cmake/StandardSettings.cmake` file. For certain options additional configuration may be needed in their respective `*.cmake` files (i.e. Conan needs the `CONAN_REQUIRES` and might need the `CONAN_OPTIONS` to be setup for it work correctly; the two are set in the `cmake/Conan.cmake` file).

11.4.4.3 Generating the documentation

In order to generate documentation for the project, you need to configure the build to use Doxygen. This is easily done, by modifying the workflow shown above as follows:

```
mkdir build/ && cd build/
cmake .. -D<project_name>_ENABLE_DOXYGEN=1 -DCMAKE_INSTALL_PREFIX=/absolute/path/to/
↳ custom/install/directory
cmake --build . --target doxygen-docs
```

Note: This will generate a `docs\` directory in the `**project's root directory**`.

11.4.4.4 Running tests

By default, the template uses [Google Test](#) for unit testing. Unit testing can be disabled in the options, by setting the `ENABLE_UNIT_TESTING` (from `cmake/StandardSettings.cmake`) to be false. To run the tests, simply use `CTest`, from the build directory, passing the desired configuration for which to run tests for. An example of this procedure is:

```
cd build          # if not in the build directory already
ctest -C Release  # or `ctest -C Debug` or any other configuration you wish to test

# you can also run tests with the `-VV` flag for a more verbose output (i.e.
#GoogleTest output as well)
```

11.4.5 FAQ

None yet.

11.5 pub-thesis-latex

11.5.1 Purpose

pub-thesis is a latex based template designed for University theses. It is especially suited for Bachelor-, Master theses and dissertations.

The [CUED PhD thesis template](#) served as basis for this template.

11.5.2 Design

pub-thesis is a modular latex template, which is reflected in the folder structure. The main tex files are `thesis.tex` and `thesis-info.tex`.

`thesis-info.tex` mostly defines general information such as name, degree, university etc and `thesis.tex` includes all other tex files such as abstracts, chapters etc.

The tex files for these chapters are found in their respective subfolders.

All figures go inside the `Figs` subfolder and all references should be included in `References/references.bib`.



(continues on next page)

(continued from previous page)

```

├── WallE.png
├── Vector
│   ├── minion.eps
│   ├── TomandJerry.eps
│   └── WallE.eps
├── Chapter3
│   └── chapter3.tex
├── compile-thesis.sh
├── compile-thesis-windows.bat
├── cookietemple.cfg
├── .cookietemple.yml
├── Declaration
│   └── declaration.tex
├── Dedication
│   └── dedication.tex
├── Dockerfile
├── Figs
│   ├── CollegeShields
│   │   ├── Downing.eps
│   │   ├── Downing.pdf
│   │   ├── Fitzwilliam.eps
│   │   ├── Fitzwilliam.pdf
│   │   ├── FitzwilliamRed.eps
│   │   ├── FitzwilliamRed.pdf
│   │   ├── Gonville_and_Caius.jpg
│   │   ├── Kings.eps
│   │   ├── Kings.pdf
│   │   ├── Licenses.md
│   │   ├── Peterhouse.pdf
│   │   ├── Queens.eps
│   │   ├── Queens.pdf
│   │   ├── src
│   │   │   ├── Downing.svg
│   │   │   ├── Kings.svg
│   │   │   ├── Peterhouse.svg
│   │   │   ├── Queens.svg
│   │   │   └── Trinity.svg
│   │   ├── StJohns.eps
│   │   ├── StJohns.pdf
│   │   ├── Trinity.eps
│   │   └── Trinity.pdf
│   ├── University_Crest.eps
│   ├── University_Crest_Long.eps
│   ├── University_Crest_Long.pdf
│   └── University_Crest.pdf
├── .github
│   └── workflows
│       └── build_thesis.yml
├── .gitignore
├── glyphtounicode.tex
├── hooks
│   └── install.sh

```

(continues on next page)

(continued from previous page)

```
└─ pre-commit
└─ LICENSE
└─ Makefile
└─ PhDThesisPSnPDF.cls
└─ Preamble
    └─ preamble.tex
└─ README.rst
└─ References
    └─ references.bib
└─ sty
    └─ breakurl.sty
└─ thesis-info.tex
└─ thesis.pdf
└─ thesis.ps
└─ thesis.tex
└─ Variables.ini
```

11.5.3 Included frameworks/libraries

1. LaTeX, XeLaTeX and LuaLaTeX support
2. Draft mode: Draft water mark, timestamp, version numbering and line numbering
3. Bibtex support
4. A Github workflow `build_thesis.yml`, which builds your thesis in a Docker container

11.5.4 Usage

11.5.4.1 Building your thesis - LaTeX / PDFLaTeX

11.5.4.1.1 Using latexmk (Unix/Linux/Windows)

This template supports `latexmk`. To generate DVI, PS and PDF run

```
latexmk -dvi -ps -pdf thesis.tex
```

11.5.4.1.2 Using the make file (Unix/Linux)

The template supports PDF, DVI and PS formats. All three formats can be generated with the provided `Makefile`.

To build the PDF version of your thesis, run:

```
make
```

This build procedure uses `pdflatex` with `bibtex` and will produce `thesis.pdf`. To use `pdflatex` with `biblatex`, you should run:

```
make BIB_STRATEGY=biblatex
```

To use XeLaTeX, you should run:

```
make BUILD_STRATEGY=xelatex
```

or with biblatex

```
make BUILD_STRATEGY=xelatex BIB_STRATEGY=biblatex
```

To use LuaLaTeX, you should run:

```
make BUILD_STRATEGY=lualatex
```

or with biblatex

```
make BUILD_STRATEGY=lualatex BIB_STRATEGY=biblatex
```

To produce DVI and PS versions of your document, you should run:

```
make BUILD_STRATEGY=latex
```

This will use the `latex` command to build the document and will produce `thesis.dvi`, `thesis.ps` and `thesis.pdf` documents. You will need `psutils` installed

Clean unwanted files

To clean unwanted clutter (all LaTeX auto-generated files), run:

```
make clean
```

Note: the Makefile itself is taken from and maintained at [here](#).

11.5.4.1.3 Shell script for PDFLaTeX (Unix/Linux)

Usage: `sh ./compile-thesis.sh [OPTIONS] [filename]`

[option] `compile`: Compiles the PhD Thesis

[option] `clean`: removes temporary files - no filename required

11.5.4.1.4 Using the batch file on Windows OS (PDFLaTeX)

- Open command prompt and navigate to the directory with the tex file. Run:
`compile-thesis-windows.bat`.
- Alternatively, double click on `compile-thesis-windows.bat`

11.5.4.2 Building your thesis - XeLaTeX

11.5.4.2.1 Using latexmk (Unix/Linux/Windows)

This template supports XeLaTeX compilation chain. To generate PDF run

```
latexmk -xelatex thesis.tex
makeindex thesis.nlo -s nomencl.ist -o thesis.nls
latexmk -xelatex -g thesis.tex
```

11.5.4.3 Building your thesis - LuaLaTeX

11.5.4.3.1 Using latexmk (Unix/Linux/Windows)

This template supports LuaLaTeX compilation chain. To generate PDF run

```
latexmk -pdflatex=lualatex -pdf thesis.tex
```

11.5.4.4 Usage details

Thesis information such as title, author, year, degree, etc., and other meta-data can be modified in `thesis-info.tex`

11.5.4.4.1 Class options

The class file, `PhDThesisPSnPDF`, is based on the standard `book` class

It supports the following custom options in the `documentclass` in `thesis.tex`:

(Usage `\documentclass[a4paper,11pt,print]{PhDThesisPSnPDF}`)

- `a4paper` (default as per the University guidelines) or `a5paper`: Paper size
- `11pt` or `12pt`: The University of Cambridge guidelines recommend using a minimum font size of 11pt (12pt is preferred) and 10pt for footnotes. This template also supports `10pt`.
- `oneside` or `twoside` (default): This is especially useful for printing double side (twoside) or single side.
- `print`: Supports Print and Online Version with different page margins and hyperlink styles. Use `print` in the options to activate Print Version with appropriate margins and page layout and view styles. Leaving the options field blank will activate Online version.
- `custommargin`: You can alter the margin dimension for both print and online version by using the keyword `custommargin` in the options. Then you can define the dimensions of the margin in the `preamble.tex` file:

```
\ifsetCustomMargin
  \RequirePackage[left=37mm,right=30mm,top=35mm,bottom=30mm]{geometry}
  \setFancyHdr
\fi
```

`\setFancyHdr` should be called when using custom margins for proper header/footer dimensions

`\ifsetMargin` is deprecated, please use `\ifsetCustomMargin` instead.

- `index`: Including this option builds the index, which is placed at the end of the thesis.
Instructions on how to use the index can be found [here](#).
Note: the package `makeidx` is used to create the index.
- `abstract`: This option enables only the thesis title page and the abstract with title and author to be printed.
- `chapter`: This option enables only the specified chapter and it's references. Useful for review and corrections.
- `draft`: The default draft mode supports some special features such as line numbers, images, and water mark with timestamp and custom text. Position of the text can be modified in `preamble.tex`.
- `draftclassic`: This mode is similar to the default draft mode in the `book` class. Images are not loaded.
- `lineno`: Enables pagewise line numbering on the outer edge. You can switch-off line numbering by specifying `nolineno` in the options.

- `flushleft`: The University recommends using ragged right or flush left alignment for texts. The reason behind this is left justifying a text may exclude a certain readers. Dyslexic people find it hard to read justified text. You can enable `raggedright` option in the document class by passing `flushleft` argument. Default is flush left and right.

11.5.4.4.2 Title page

The front page (title page) resizes to fit your title length. You can modify the options in `thesis-info.tex`.

- `\subtitle` (optional): Adds a subtitle to your thesis.
- `\college` (optional): This option adds the name of your college on the bottom left.

If `\college` is defined, the bottom of the title page will look like this:

King's College	2014
----------------	------

If `\college` is undefined or blank, the `degreedate` will be centered.

2014

The template offers support to having both the college and university crests or just one of the crests.

- `\collegeshield` (optional): Includes college crest in addition to the university crest. This reformats the front page layout.

11.5.4.4.3 Abstract separate

- A separate abstract with the title of the PhD and the candidate name has to be submitted to the Student Registry. This can be generated using `abstract` option in the document class. Ignore subsequent warnings about skipping sections (if any).
- To generate the separate abstract and the title page, make sure the following commands are in the preamble section of `thesis.tex` file:

```
\ifdefineAbstract
\includeonly{Abstract/abstract}
\fi
```

11.5.4.4.4 Chapter mode

- The chapter mode allows user to only print specific chapters along with references. By default, it excludes everything else in the front matter and appendices. This can be done by using `chapter` option in the document class in `thesis.tex`. Ignore subsequent warnings about skipping sections (if any).
- To generate the separate abstract and the title page, make sure the following commands are in the preamble section of `thesis.tex` file:

```
\ifdefineChapter
\includeonly{Chapter3/chapter3}
\fi
```

11.5.4.4.5 Draft

`draft` adds a watermark `draft` text with timestamp and version number at the top or the bottom of the page. Pagewise line numbering is added on every page. `draft` settings can be tweaked in the `preamble.tex`.

- Use `draftclassic` in the document class options to use the default book class draft mode.
- To add figures in draft mode (default enabled), in the preamble set `\setkeys{Gin}{draft=false}`. `draft=true` disables figures
- To change the watermark text
- To change the position of the watermark text. Default watermark position is top. The location can be changed to (top / bottom)
- To change the draft version. Default draft version is v1.0.
- Watermark grayscale value can be modified. Text grayscale value (should be between 0-black and 1-white). Default value is 0.75

11.5.4.4.6 Choosing the fonts

PhDThesisPSnPDF currently supports three fonts Times, Fourier and Latin Modern (default).

- `times`: (The University of Cambridge guidelines recommend using Times). Specifying times option in the document class will use `mathptpx` or Times font with Math Support.
- `fourier`: fourier font with math support
- `default` (empty): When no font is specified, Latin Modern is used as the default font with Math Support.
- `customfont`: Any custom font can be set in preamble by using `customfont` option in the document class. Then the custom font can be loaded in `preamble.tex` in the line:

```
\ifsetCustomFont
  \RequirePackage{Your_Custom_Font}
\fi
```

11.5.4.4.7 Choosing the bibliography style

PhDThesisPSnPDF currently supports two styles `authoryear` and `numbered` (default). Citation style has to be set. You can also specify `custombib` style and customise the bibliography.

- `authoryear`: For author-year citation eg., Krishna (2013)
- `numbered`: (Default Option) For numbered and sorted citation e.g., [1,5,2]
- `custombib`: Define your own bibliography style in the `preamble.tex` file.

```
\RequirePackage[square, sort, numbers, authoryear]{natbib}
```

- (Overview of Bibtex-Styles with preview)[<http://nodonn.tipido.net/bibstyle.php?>]
- If you would like to use `biblatex` instead of `natbib`. Pass the option `custombib` in the documentclass. In the `preamble.tex` file, edit the `custombib` section. Make sure you don't load the `natbib` package and you can specify the layout of your references in `thesis.tex` in the reference section. If you are using `biber` as backend, run `pdflatex thesis.tex >> biber thesis >> pdflatex thesis.tex >> biber thesis >> pdflatex thesis.tex`. If you are using the default `natbib` package, don't worry about this.

11.5.4.4.8 Choosing the page style

PhDThesisPSnPDF defines 3 different page styles (header and footer). The following definition is for twoside layout. To choose a page style, include it in the documentclass options: `\documentclass[PageStyleI]{PhDThesisPSnPDF}`. Alternatively, page style can be changed by adding `\pagestyle{PageStyleI}` or `\pagestyle{PageStyleII}` in `thesis.tex`. Note: Using `\pagestyle` command will override documentclass options when used globally.

- `default` (leave empty): For Page Numbers in Header (Left Even, Right Odd) and Chapter Name in Header (Right Even) and Section #. Section Name (Left Odd). Blank Footer.

Header (Even)	: 4	Introduction
Header (Odd)	: 1.2 Section Name	5
Footer	: Empty	

- `PageStyleI`: For Page Numbers in Header (Left Even, Right Odd) and Chapter Name next to the Page Number on Even Side (Left Even). Section Number and Section Name and Page Number in Header on Odd Side (Right Odd). Footer is empty. Layout:

Header (Even)	: 4 Introduction
Header (Odd)	: 1.2 Section Name 5
Footer	: Empty

- `PageStyleII`: Chapter Name on Even Side (Left Even) in Header. Section Number and Section Name in Header on Odd Side (Right Odd). Page numbering in footer. Layout:

Header (Even)	: Introduction
Header (Odd)	: 1.2 Section Name
Footer[centered]:	3

11.5.4.4.9 Changing the visual style of chapter headings

The visual style of chapter headings can be modified using the `titlesec` package. Edit the following lines in the `preamble.tex` file.

```
\RequirePackage{titlesec}
\newcommand{\PreContentTitleFormat}{\titleformat{\chapter}[display]{\scshape\Large}
{\Large\filleft{\chaptertitlename} \Huge\thechapter}
{1ex}}
[\vspace{1ex}\titlerule]}
\newcommand{\ContentTitleFormat}{\titleformat{\chapter}[display]{\scshape\huge}
{\Large\filleft{\chaptertitlename} \Huge\thechapter}{1ex}
{\titlerule\vspace{1ex}\filright}
[\vspace{1ex}\titlerule]}
\newcommand{\PostContentTitleFormat}{\PreContentTitleFormat}
\PreContentTitleFormat
```

11.5.4.4.10 Custom settings

- The depth for the table of contents can be set using:

```
\setcounter{secnumdepth}{3}
\setcounter{tocdepth}{3}
```

A depth of [3] indicates to a level of \subsubsection or ###. Default set as 2.

- To hide sections from appearing in TOC use: \tohide\section{Section name} in your TeX files
- Define custom caption style for figure and table caption in preamble.tex using:

```
\RequirePackage[small,bf,figurename=Fig.,labelsep=space,tableposition=top]{caption}
```

- Uncomment the following lines in preamble.tex to force a figure to be displayed in a particular location. Use H when including graphics. Note H instead of h.

```
\usepackage{float}
\restylefloat{figure}
```

- Bibliography with Author-Year Citation in preamble.tex:

```
\RequirePackage[round, sort, numbers, authoryear]{natbib}
```

- Line spacing for the entire document can be specified in preamble.tex. Uncomment the line spacing you prefer. e.g.,

11.5.4.4.11 Nomenclature definition

- To use nomenclature in your chapters:

```
\nomenclature[g-pi]{\pi}{\simeq 3.14\ldots}
```

The sort keys have prefix. In this case a prefix of g is used to denote Greek Symbols, followed by -pi or -sort_key. Use a - to separate sort key from the prefixes. The standard prefixes defined in this class are:

- A or a: Roman Symbols
- G or g: Greek Symbols
- Z or z: Acronyms/Abbreviations
- R or r: Superscripts
- S or s: Subscripts
- X or x: Other Symbols

- You can change the Title of Nomenclature to Notations or Symbols in the preamble.tex using:

```
\renewcommand\nomname{Symbols}
```

TexStudio's default compile option doesn't include nomenclature, to compile your document with the nomenclature, do the following:

```
Options >> Configure TexStudio >> Build >> User Commands >> add user command
```

In add user command type makenomenclature:makenomenclature on the left pane and makeindex %.nlo -s nomencl.list -o %.nls on the execution side. Now you can run the user defined command makenomenclature from Tools >> User >> makenomenclature.

Alternatively, you can use the compile-thesis-windows.bat file or run make on Unix / Linux / MacOS

11.5.4.5 Git hooks

You rarely want to commit changes to your TeX files which are not reflected in the PDF included in the repo. You can automate this process, among other things, with a git hook. Install the hook with make hooks (or see how to do it in ./hooks/install.sh). Now every time you commit, if any files affecting your build have changed in this commit and those changes are more recent than the last modification of thesis.pdf, the default make target will be run and changes to thesis.pdf will be git added.

Currently, changes to any tex/pdf/eps/png/cls files are picked up. This can be changed in ./hooks/pre-commit.

Skip the hook with git commit --no-verify.

bash-only.

11.5.4.6 General guidelines

- To restrict the length of the figure caption in List of figures use a [short-title] and {longtitle} for the caption or the section:

```
\caption[Caption that you want to appear in TOC]{Actual caption of the figure}`
\section[short]{title}`
```

- To exclude sections from being numbered and disable it from appearing in the Table of Contents use or
- To only exclude it from being listed in the Table of Contents encapsulate the section command inside the \tochide command. \tochide{\section{Section_Name}} the section will not appear in the Table of Contents, but the section will be numbered.
- When including figures in your tex file, it's a good practice to size your picture depending on the page size, instead of using absolute values. In the following example 0.75\textwidth refers to picture width being set to 75% of the text width.

```
\includegraphics[width=0.75\textwidth]{minion}
```

- Use a - to separate sort key from the prefixes, eg., g-pi denotes the Greek symbol pi.

11.6 web-website-python

11.6.1 Purpose

This template is a [Flask](#) based Web Template that can be customized from two basic layouts and many available frontend templates. It contains all the code, necessary for project setup and automatic deployment on a Linux server. It also provides a GitHub Workflow for automatic CSS linting on push using [Stylelint](#).

11.6.2 Design

The whole template is designed to be as customizable as possible. Note that all templates could be customized with a full featured Frontend template setup during the template creation process. However, if you don't like the offered templates or simply want to create your own frontend, you can create your template with only a minimal frontend. You can choose from two main options:

11.6.2.1 The basic setup

The basic theme is designed to provide only minimal code needed for getting started: Thus it comes with only minimal HTML/CSS/JS code (but you can initialize it with a full featured frontend, if you want to) and basic Flask configuration. However, it contains all the code needed for automatic deployment on a Linux server and adheres to the cookiетemplate project structure standards.

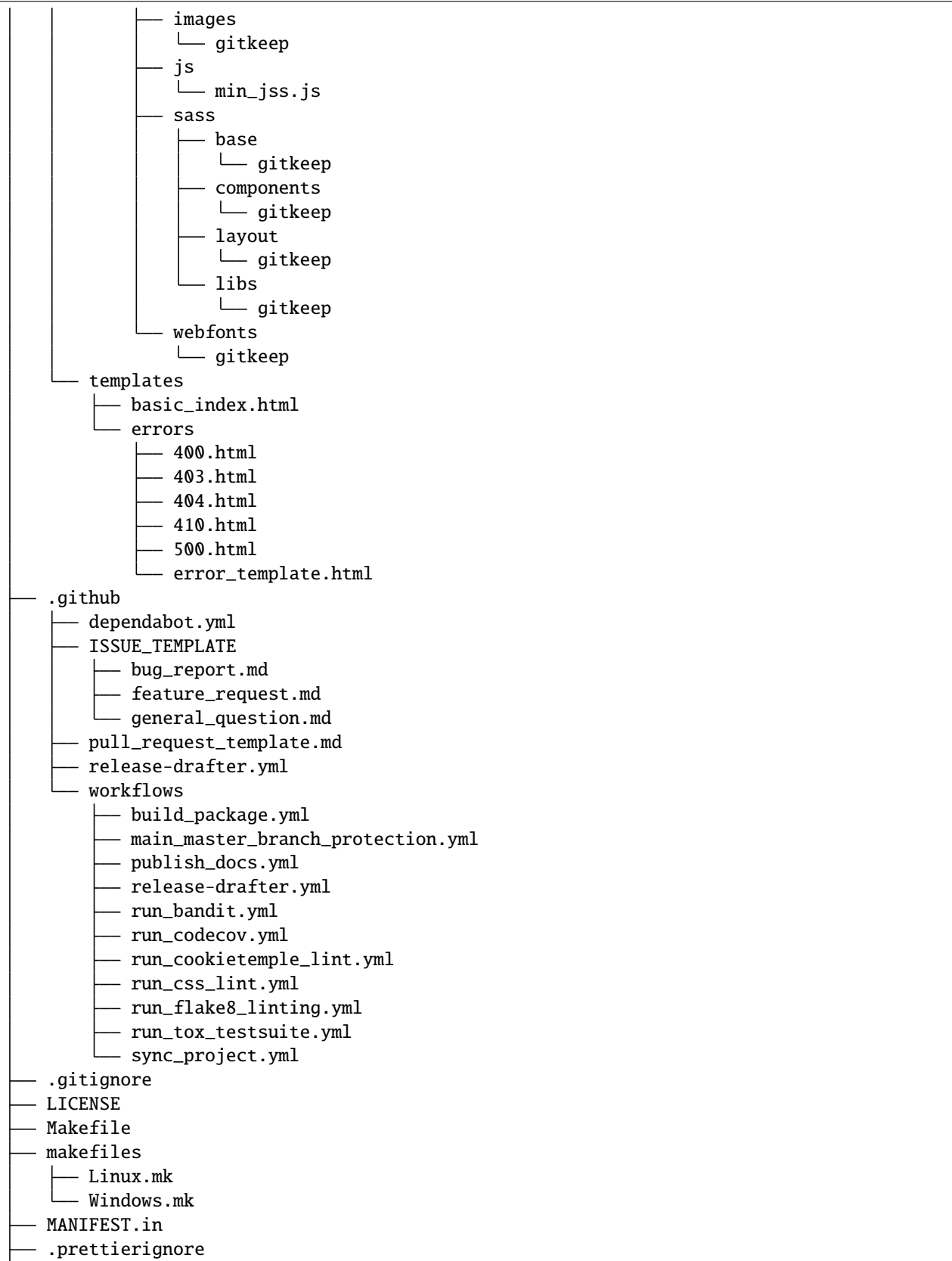
```

— .bandit.yml
— CODE_OF_CONDUCT.rst
— cookiетemplate.cfg
— .cookiетemplate.yml
— deployment_scripts
  — exploding_springfield
  — exploding_springfield.service
  — README.md
  — setup.sh
— Dockerfile
— docs
  — authors.rst
  — code_of_conduct.rst
  — conf.py
  — index.rst
  — installation.rst
  — make.bat
  — Makefile
  — readme.rst
  — requirements.txt
  — _static
    — custom_cookiетemplate.css
  — usage.rst
— .editorconfig
— exploding_springfield
  — app.py
  — basic
    — __init__.py
    — routes.py
  — config.py
  — errors
    — handlers.py
    — __init__.py
  — __init__.py
  — server.py
  — static
    — assets
      — css
        — min_css.css

```

(continues on next page)

(continued from previous page)



(continues on next page)

(continued from previous page)

```

— README.rst
— .readthedocs.yml
— requirements_dev.txt
— requirements.txt
— setup.cfg
— setup.py
— .stylelintrc.json
— tests
  — __init__.py
  — test_exploding_springfield.py
— tox.ini

```

11.6.2.2 The advanced setup

The advanced theme comes with a lot more functionality by default (and can also be initialized with a full featured, nice frontend):

1. It uses [FlaskSQLAlchemy](#) and [FlaskMigrate](#) to setup a [SQLite](#) application for simple User Login.
2. It provides translation for German and English using [Flask-Babel](#).
3. It provides sending mail through [Flask-Mail](#).
4. It provides error handling through custom error pages.
5. Its configured to be automatically deployed in seconds on a Linux server.
6. More is WIP (Contributions are welcome).

```

— babel.cfg
— .bandit.yml
— CODE_OF_CONDUCT.rst
— cookietemple.cfg
— .cookietemple.yml
— deployment_scripts
  — exploding_springfield
  — exploding_springfield.service
  — README.md
  — setup.sh
— Dockerfile
— docs
  — authors.rst
  — code_of_conduct.rst
  — conf.py
  — index.rst
  — installation.rst
  — make.bat
  — Makefile
  — readme.rst
  — requirements.txt
  — _static
    — custom_cookietemple.css
  — usage.rst
— .editorconfig

```

(continues on next page)

(continued from previous page)

```

— exploding_springfield
  — app.py
  — auth
    — forms
      — __init__.py
      — login_form.py
      — register_form.py
    — __init__.py
    — routes.py
  — config.py
  — errors
    — handlers.py
    — __init__.py
  — __init__.py
  — main
    — __init__.py
    — routes.py
  — models
    — __init__.py
    — users.py
  — server.py
  — services
    — __init__.py
  — static
    — assets
      — css
        — min_css.css
      — images
        — gitkeep
      — js
        — min_jss.js
      — sass
        — base
          — gitkeep
        — components
          — gitkeep
        — layout
          — gitkeep
        — libs
          — gitkeep
      — webfonts
        — gitkeep
    — mail_stub.conf
  — templates
    — auth
      — login.html
      — register.html
    — base.html
    — errors
      — 400.html
      — 403.html
      — 404.html

```

(continues on next page)

(continued from previous page)



11.6.3 Included frameworks/libraries

Both templates are based on [Flask](#) and, in the case of the advanced layout, make heavy use of its extensions.

1. [Flask](#)
2. [click](#), [argparse](#) or no command line interface
3. [pytest](#) or [unittest](#) as testing frameworks
4. Preconfigured [tox](#) to run [pytest](#) matrices with different Python environments
5. Preconfigured [readthedocs](#)
6. Eleven Github workflows:
 - └─ [build_package.yml](#)
 - └─ [main_master_branch_protection.yml](#) | └─ [publish_docs.yml](#) | └─ [release-drafter.yml](#) | └─ [run_bandit.yml](#) | └─ [run_codecov.yml](#) | └─ [run_cookietemplate_lint.yml](#)
 - └─ [run_css_lint.yml](#) | └─ [run_flake8_linting.yml](#) | └─ [run_tox_testsuite.yml](#) | └─ [sync_project.yml](#)
 1. [publish_docs.yml](#), which builds and publishes the [readthedocs](#) documentation.
 2. [build_package.yml](#), which builds the web-template package.
 3. [run_flake8_linting.yml](#), which runs [flake8](#) linting.
 4. [run_tox_testsuite.yml](#), which runs the [tox](#) testing suite.
 5. [run_css_lint.yml](#), which runs [Stylelint](#) CSS linting.
 6. [run_codecov](#), apply [codecov](#) to your project/PRs in your project and create automatically a report with the details at [codecov.io](#)
 7. [run_bandit](#), run [bandit](#) to discover security issues in your python code
 8. [main_master_branch_protection](#): Please read [main_master_branch_protection workflow](#).
 9. [release-drafter.yml](#): Please read [release drafter workflow](#).
 10. [run_cookietemplate_lint.yml](#), which runs `cookiетemplate lint` on the project.
 11. [sync_project.yml](#), which syncs the project to the most recent `cookiетemplate` template version

We highly recommend to use `click` (if commandline interface is required) together with `pytest`.

The advanced template therefore uses some more packages including:

1. [FlaskSQLAlchemy](#)
2. [Flask-Migrate](#)
3. [Flask-Babel](#) for translations
4. [Flask-Mail](#) for mail
5. [Flask-Bootstrap](#) for basic login page styling
6. [Flask-Login](#) for login session management
7. [Flask-wtf](#) for the login forms

11.6.4 Usage

11.6.4.1 The basic template usage

The generated flask web project can be installed using:

```
$ make install
```

or alternatively:

```
$ python setup.py install
```

Your package is then installed globally (or in your virtual environment) on your machine and can be called from your favorite shell:

```
$ <<your_project_name>>
```

Other make targets include:

```
$ make clean
```

which removes all build files:

```
$ make dist
```

which builds source and wheel packages, which can then be used for a PyPi release using:

```
$ make release
```

All possible Makefile commands can be viewed using:

```
$ make help
```

Another possibility is to simply run:

```
$ export FLASK_APP = path/to/your/app.py  
$ flask run
```

Note that, if your current directory contains your app.py file, you do not need to set the environment variable lika above!

11.6.4.2 The advanced template usage

Using the advanced template, you have to consider a few more steps in order to make it work properly:

1. You can install the project just like described above via `$ make install`.
2. Now, you have to setup and initialize your SQLite database file using `$ make init_db`. This step is needed otherwise your app won't work!
3. In order to make your translations working, we need to update and compile the recent translations Therefore `$ flask translate update` and then `$ flask translate compile`. Note that you have to `$ export FLASK_APP=your/path/to/app.py` if not already done. Then, again, run `$ make install` to pick up your translations into your actual build.
3. Now, fire up `$ <<your_project_name>>` and see your project setup working.

A quick note on translations: Your advanced template comes with a basic translation setup for German and English translation. As your project grows, you may need to add new translations. This can be easily done using the provided cli-commands by the template:

1. If you want to add a new language: Use `$ flask translate init <<my_new_language>>`. Note that my new language must be a valid language literal like `en` for english.
2. `$ flask translate update` to update all language repositories
3. Now you can update your translations in `your/path/to/translations/yourlanguage/LC_MESSAGES/messages.po`.
3. `$ flask translate compile` to compile all language repositories

Note that you need to run `$ make install` each time after updating and compiling your new translations in order for them to take effect. However, this is not necessary, if you start your application via `$ flask run`.

11.6.5 Automatic Deployment

IMPORTANT: Note that the following is written for a server running Ubuntu 18.04 LTS where Python2 is still the default. If you are using Ubuntu 20 (or similar), you can replace `pip3` with `pip` and `python3` with `python`.

Both templates are ready for deployment using nginx and gunicorn and are therefore shipped with a setup script `path/to/your/project/deployment_scripts/setup.sh`. There are a few requirements needed in order to deploy:

1. You need a registered Domain from your preferred DNS-Provider like [Namecheap](#).
2. You need a Linux server, like a droplet at [DigitalOcean](#), in order to deploy your application.
3. To start deployment, you have to setup your server initially. You can follow, for example, the steps [here](#) in order to correctly setup your server.

If you meet all the requirements above login (for example via `$ ssh yourvmusername@your-servers-IP`) into your server:

Now, you need to clone your repository in order to start the deployment process. So `$ git clone <<GITHUB_URL_OF_YOUR_PROJECT>>` and `cd $ YOUR_PROJECTS_TOP_LEVEL_DIRECTORY`. Now simply run `$ source deployment_scripts/setup.sh` and the deployment starts. You may be prompted for your password as some commands run need sudo rights.

Important: Currently, one more step is required to get https redirecting to work properly. This will be included into a script in the future, to automate this process.

1. `$ sudo vim /etc/nginx/sites-enabled/<<my_project_name>>`
2. Now, you need to copy the certbot added section from the second server section into the first server section, so copy: `listen 443 ssl; # managed by Certbot ssl_certificate /etc/letsencrypt/live/<<my_url>>/fullchain.pem; # managed by Certbot ssl_certificate_key /etc/letsencrypt/live/<<my_url>>/privkey.pem; # managed by Certbot include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot`

into the first server section after the location and delete it from the second one.

3. `$ sudo nginx -t`
4. `$ sudo nginx -s reload`
5. `$ sudo systemctl restart <<my_project_name>>`

Tip: You can check `$ sudo systemctl status <my_project_name>` to check for the working state of your gunicorn instance or any errors.

If everything went fine, you should now be able to access your application at your domain. Note that the setup process also includes HTTP to HTTPS redirecting.

In case of any problems, dont hesitate to drop us a message in our [Discord](#). or create an issue [at our github repo](#)

11.6.6 FAQ

None yet.

11.7 Shared FAQ

11.7.1 What are the available domains?

cookietemplate currently offers a total of 5 different template domains:

1. cli
2. lib
3. gui
4. web
5. pub

Note that the lib domain does not offer a Python template. We recommend the usage of the cli-python template which can easily be adapted for such purposes by removing the boilerplate command line code.

11.7.2 How do I publish my documentation?

cookietemplate ships with a full, production ready [Read the Docs](#) setup and with a complete gh-pages setup.

11.7.2.1 Read the Docs

You need to [import your documentation](#) on Read the Docs website. Do not forget to sync your account first to see your repository. Your documentation will then be available on `https://repositoryname.readthedocs.io/`

11.7.2.2 Github Pages

Your documentation is automatically pushed to the gh-pages branch. Follow the documentation on [configuring a publishing source for your Github pages site](#) <`https://docs.github.com/en/free-pro-team@latest/github/working-with-github-pages/configuring-a-publishing-source-for-your-github-pages-site`>_ and select the gh-pages branch. Your documentation will then be available on `https://username.github.io/repositoryname`.

11.7.3 What is Dependabot and how do I set it up?

Dependabot is a service, which (for supported languages) automatically submits pull requests for dependency updates. cookietemple templates ship with dependabot configurations, if the language is supported by Dependabot. To enable Dependabot you need to login (with your Github account) and add your repository (or enable Dependabot for all repositories). Note that you need to do this for every organization separately. Dependabot will then pick up the configuration and start submitting pull requests!

11.7.4 Release Drafter

In Cookietemple 1.3.0, all templates received a new GitHub Action called Release drafter. This replaces the CHANGELOG.rst file of all templates. Release drafter automatically includes references to all PRs made to the default branch and a description of what this PR was about (basically the PRs title) in a release draft on GitHub. Per default, the release drafter of cookietemple's templates has been configured to distinguish between two main categories: Features and Fixes. Every PR that was made from a branch called either feature/ (for Features) or fix/ (for Fixes) will automatically be grouped into those categories and be labelled automatically. If no category was found, the changes will be grouped into a common Changes category. See also `_release_drafter_workflow`.

11.7.5 How do I add a new template?

Please follow *[Adding new templates](#)*.

GITHUB SUPPORT

12.1 Overview

cookiecutter uses [GitPython](#) and [PyGithub](#) to automatically create a repository, add, commit and push all files. Moreover, issue labels, a development and a TEMPLATE branch are created. The TEMPLATE branch is required for *Syncing your project* to work and should not be touched manually.

12.2 Branches

12.2.1 Overview

git branches can be understood as diverging copies of the main line of development and facilitate parallel development. To learn more about branches read [Branches in a Nutshell](#) of the [Pro Git Book](#). A simple best practice development workflow follows the pattern that the master/main branch always contains the latest released code. It should only be touched for new releases. Code on the master/main branch must compile and be as bug free as possible. Development takes place on the development branch. All in parallel developed features eventually make it into this branch. The development branch should always compile, but it may contain incomplete features or known bugs. cookiecutter creates a TEMPLATE branch, which is required for *Syncing your project* to work and should not be touched manually.

12.2.2 Branch protection rules

cookiecutter sets several branch protection rules, which enforce a minimum standard of best branch practices. For more information please read [about protected branches](#). The following branch protection rules only apply to the master/main branch:

1. Required review for pull requests: A pull request to master/main can only be merged if the code was at least reviewed by one person. If you are developing alone you can merge with your administrator powers.
2. Dismiss stale pull request approvals when new commits are pushed.

12.3 Github Actions

12.3.1 Overview

Modern development tries to merge new features and bug fixes as soon as possible into the `development` branch, since big, diverging branches are more likely to lead to merge conflicts. This practice is known as [continuous integration](#) (CI). Continuous integration is usually complemented with automated tests and continuous delivery (CD). All of cookietemple's templates feature [Github Actions](#) as main CI/CD service. Please read the [Github Actions Overview](#) for more information. On specific conditions (usually push events), the Github Actions workflows are triggered and executed. The developers should ensure that all workflows always pass before merging, since they ensure that the package still builds and all tests are executed successfully.

12.3.2 `main_master_branch_protection` workflow

All templates feature `main_master_branch_protection` workflow. This workflow runs everytime a PR to your projects master or main branch is created. It fails, if the PR to the master/main branch origins from a branch that does not contain `patch` or `release` in its branch name. If development code is written on a branch called `development` and a new release of the project is to be made, one should create a `release` branch only for this purpose and then merge it into master/main branch. This ensures that new developments can already be merged into `development`, while the release is finally prepared. The `patch` branch should be used for required hotfixes (checked out directly from master/main branch) because, in the meantime, there might multiple developments going on at `development` branch and you dont want to interfere with them. Pull requests against the master or main branch should not contain any SNAPSHOT versions, since they are only used for development versions.

12.3.3 `release drafter` workflow

All templates feature `release-drafter` workflow. This workflow consists of two parts: Everytime a new PR is made, the workflow runs and tries autolabeling the PR either as `feature` or `bug`. Feature PRs introduce new features if the branch name contains "feature". Bug PRs are PRs that either have a title containing "fix" or the branch name contains "fix". This Action then drafts a new release grouped by the different PR categories and include references and titles to all PRs included in the new release. One can read more about this at [the Release drafter GitHub repo](#).

12.3.4 `sync_project.yml`

All templates also feature this workflow. This workflow is used for automatic syncing (if enabled) your project with the latest cookietemple template version. The workflow is run every night, although this behavior can be customized if desired. The workflow calls `cookietemple sync`, which first checks whether a new template version is available and if so it submits a pull request. For more details please visit [Syncing your project](#).

12.4 Secrets

Github secrets are what their name suggests: Encrypted secret values in a repository or an organisation; once they are set their value can be used for sensible data in a project or an organisation but their raw value can never be seen again even by an administrator (but it can be updated).

Cookietemplate uses a secret called CT_SYNC_TOKEN for its syncing feature. This secret is automatically created during the repo creation process, if you choose to create a GitHub repo. The secret contains your encrypted personal access token as its value. Note that this will have no effect on how to login or any other activity in your project. If you remove the secret or change its value (even with another personal access token of you) the syncing feature will no longer work. In case you are creating an organisation repository, the secret will also be stored as a repository secret, only usable for your specific project.

See section below in case your Github repo creation failed during the create process.

12.4.1 Error Handling due to failed Github repository creation

Errors during the create process due to a failed Github repo creation may occur due to a vast amount of reasons: Some common error sources are:

1. You have no active internet connection or your firewall protects you against making calls to external APIs.
2. The Github API service or Github itself is unreachable at the moment, which can happen from time to time. In doubt, make sure to check [the Github status page](#).
3. A repo with the same name already exists in your account/your organisation.
4. Your Github Token/Secret does not have all required permissions (all repository and workflow permissions).

Creation fails, ok: But how can I then access the full features of cookietemplate? You can try to fix the issue (or wait some time on case, for example, when Github is down) and then process to create a Github repository manually. After this, make sure to create a secret named CT_SYNC_TOKEN with the value of your PAT for your repository. See [the Github docs](#) for more information on how to create a secret.

We're planning to provide a command like `cookiетemplate config fix-github` that tries to create a Github repo, set the secret and all other stuff that is going on during the Github repository creation in the create process in a later version.

12.5 Issue labels

cookiетemplate's Github support automatically creates [issue labels](#). Currently the following labels are automatically created: 1. dependabot: All templates, which include [Dependabot](#) support label all Dependabot pull requests with this label.

CONTRIBUTING

Contributions are welcome and greatly appreciated! Every little bit helps and credit will always be given. If you have any questions or want to get in touch with the core team feel free to join our [Discord server](#).

You can contribute in many ways:

13.1 Types of Contributions

13.1.1 Report Bugs

Report bugs at <https://github.com/cookiejar/cookiecutter/issues>.

If you are reporting a bug, please:

- Use the appropriate issue template.
- Be as detailed as possible. The more time you invest into describing the bug, the more time we save solving them, effectively allowing us to improve cookiecutter at a faster pace.
- Be patient. We are passionate, hard workers, but also have demanding full time jobs, which require a lot of our attention.

13.1.2 Fix Bugs

Look through the GitHub issues for bugs. We would appreciate it if you quickly commented on the respective issue and write that you are working on this bug, to minimize the chances of two people working on the same task.

13.1.3 Implement Features

Look through the GitHub issues for features. The same rule also applies to features. Please write if you're picking up one of the feature suggestions.

13.1.4 Add Templates

If you're planning to add a new template to cookiетemplate we highly suggest that you open an issue using the corresponding template and discuss it first with us.

Adding new templates will guide you through the process of adding new templates to cookiетemplate.

Please ensure that you are following all the guidelines and that your template meets the requirements.

13.1.5 Write Documentation

cookiетemplate could always use more documentation, whether as part of the official cookiетemplate docs, in docstrings, or even on the web in blog posts, articles, and such.

13.1.6 Submit Feedback

The best way to send feedback is to file an issue [here](#).

If you are proposing a feature:

- Use the appropriate GitHub issue
- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

13.2 Get Started!

Ready to contribute? Here's how to set up *cookiетemplate* for local development.

1. Fork the *cookiетemplate* repo on GitHub.
2. Clone your fork locally

```
$ git clone git@github.com:your_name_here/cookiетemplate.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development

```
$ mkvirtualenv cookiетemplate
$ cd cookiетemplate/
$ python setup.py develop
```

4. Create a branch for local development. Note that you should always start your branch name with either *fix* or *feature* so that the Release Drafter GitHub App will handle your PR right.

```
$ git checkout -b fix_or_feature/name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass the nox test suite.

```
$ nox
```

To get nox (and nox-poetry), just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin fix_or_feature/name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

13.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated.
Put your new functionality into a function with a docstring, and add the feature to the list in README.rst, if it is a major feature.
3. The pull request should work for Python 3.7+ and for PyPy. Check your pull request on Github and verify that all checks and GitHub workflows are passing!

13.4 Tips

To run a subset of tests:

```
.. code-block:: console
```

```
$ pytest tests/something
```


ADDING NEW TEMPLATES

Adding new templates is one of the major improvements and community contributions to cookietemplate, which is why we are dedicating a whole section to it. Please note that creating new templates is a time consuming task. So be prepared to invest a few hours to bring a new template to life. The integration into cookietemplate however, is straightforward if you follow the guide below. Due to the tight coupling of our templates with all cookietemplate commands such as `create`, `list`, `info`, `lint` and `bump-version`, new templates require the modification of several files.

cookietemplate uses `cookiecutter` to create all templates. You need to familiarize yourself beforehand with cookiecutter to able to write templates, but don't worry, it's pretty easy and you usually get by with very few cookiecutter variables. You can start with your [very first cookiecutter template](#) and then simply see how the other existing cookietemplate templates are made and copy what you need.

The following sections will line out the requirements for new templates and guide you through the process of adding new templates step by step. Nevertheless, we strongly encourage you to discuss your proposed template first with us in public *via* a Github issue.

14.1 Template requirements

To keep the standard of our templates high we enforce several standards, to which all templates **must** adhere. Exceptions, where applicable, but they would have to be discussed beforehand. Hence, the term *should*.

1. New templates must be novel. We do not want a second cli-python template, but you are of course always invited to improve it. A new commandline library does not warrant an additional template, but rather modifications of the existing template with cookiecutter if statements. However, distinct modifications of already existing templates may be eligible. An example would be to add a GUI template for a language, which does not yet have a GUI template. Templates for domains, which we do not yet cover or additional languages to already existing domains are of course more than welcome.
2. All templates should be cutting edge and not be based on technical debt or obscure requirements. Our target audience are enthusiastic open source contributors and not decades old companies stuck with Python 2.7.
3. All templates should build as automatically as possible and download all dependencies without manual intervention.
4. All templates must have a testing and possibly mocking framework included.
5. All templates must provide a readthedocs setup, a README.rst, usage.rst and installation.rst file, a LICENSE, Github issue and pull request templates and a .gitignore file. Moreover, a .dependabot configuration should be present if applicable. Note that most of these are already included in our common_files and do not need to be rewritten. More on that below.
6. All templates must provide a Makefile, which wraps heavily used commands to unify common operations such as installing, testing or distributing a project.
7. All templates should have a Dockerfile, which provides an entrypoint for the project.

8. All templates must implement all required functionality to allow the application of all commands mentioned above to them, which includes a cookiетemplate.cfg file, the template being in the available_templates.yml and more.
 9. All templates must have Github workflows, which at least build the documentation and the project.
 10. Every template must also have a workflow inside cookiетemplate, which creates a project from the template with dummy values.
 11. Your template must support Linux and MacOS. Windows support is optional, but strongly encouraged.
- Again, we strongly suggest that new templates are discussed with the core team first.

14.2 Step by step guide to adding new templates

Let's assume that we are planning to add a new commandline [Brainfuck](#) template to cookiетemplate. We discussed our design at length with the core team and they approved our plan. For the sake of this tutorial **we assume that the path / always points to /cookiетemplate**. Hence, at this level we see cookiетemplate_cli.py and a folder per CLI command.

1. Let's add our brainfuck template information to /create/templates/available_templates.yml below the cli section.

```

1 cli:
2   brainfuck:
3     name: Brainfuck Commandline Tool
4     handle: cli-brainfuck
5     version: 0.0.1
6     available libraries: none
7     short description: Brainfuck Commandline tool with ANSI coloring
8     long description: Amazing brainfuck tool, which can even show pretty unicorns in
9     ↪ the console.
        Due to ANSI coloring support they can even be pink! Please someone send help.
```

2. Next, we add our brainfuck template to /create/templates

Note that it should adhere to the standards mentioned above and include all required files. Don't forget to add a cookiетemplate.cfg file to facilitate bump-version. See [Configuration](#) for details. It is **mandatory** to name the top level folder `{{ cookiecutter.project_slug }}`, which ensures that the project after creation will have a proper name. Furthermore, the cookiecutter.json file should have at least the following variables:

```

1 {
2   "full_name": "Homer Simpson",
3   "email": "homer.simpson@posteo.net",
4   "project_name": "sample-cli",
5   "project_slug": "sample-cli",
6   "version": "1.0.0",
7   "project_short_description": "Command-line utility to...",
8   "github_username": "homer_github"
9 }
```

The file tree of the template should resemble

```

1 └─ cookiecutter.json
2 └─ {{ cookiecutter.project_slug }}
3     └─ docs
```

(continues on next page)

(continued from previous page)

```

4 |   |   | installation.rst
5 |   |   | usage.rst
6 |   |   | .github
7 |   |   | | workflows
8 |   |   | |   build_brainfuck.yml
9 |   | hello.bf
10 |   | cookietemplate.cfg
11 |   | README.rst

```

3. Now it is time to subclass the TemplateCreator to implement all required functions to create our template!

Let's edit /create/domains/cli_creator.py. Note that for new domains you would simply create a new file called DomainCreator.

In this case we suggest to simply copy the code of an existing Creator and adapt it to the new domain. Your new domain may make use of other creation functions instead of create_template_without_subdomain, if they for example contain subdomains. You can examine create/TemplateCreator.py to see what's available. You may also remove functions such as the creation of common files.

If we have any brainfuck specific cookiecutter variables that we need to populate, we may add them to the TemplateStructCli.

Our brainfuck templates does not have them, so we just leave it as is.

For the next step we simply go through the CliCreator class and add our brainfuck template where required. Moreover, we implement a cli_brainfuck_options function, which we use to prompt for template specific cookiecutter variables.

Assuming cli_creator.py already contains a cli-java template

```

1 @dataclass
2 class TemplateStructCli(CookietemplateTemplateStruct):
3     """
4     Intended Use: This class holds all attributes specific for CLI projects
5     """
6
7     """ ____JAVA____ """
8     main_class_prefix: str = ''
9
10    """ ____BRAINFUCK____ """
11
12
13    class CliCreator(TemplateCreator):
14
15        def __init__(self):
16            self.cli_struct = TemplateStructCli(domain='cli')
17            super().__init__(self.cli_struct)
18            self.WD = os.path.dirname(__file__)
19            self.WD_Path = Path(self.WD)
20            self.TEMPLATES_CLI_PATH = f'{self.WD_Path.parent}/templates/cli'
21
22            """ TEMPLATE VERSIONS """
23            self.CLI_JAVA_TEMPLATE_VERSION = super().load_version('cli-java')
24            self.CLI_BRAINFUCK_TEMPLATE_VERSION = super().load_version('cli-brainfuck')
25
26        def create_template(self, path: Path, dot_cookiecutter: dict or None):
27            """

```

(continues on next page)

(continued from previous page)

```

28     Handles the CLI domain. Prompts the user for the language, general and domain
↳ specific options.
29     """
30
31     self.cli_struct.language = cookiетemplate_questionary_or_dot_cookiетemplate(function=
↳ 'select',
32                                     question=
↳ 'Choose the project\'s primary language',
33                                     choices=[
↳ 'python', 'java', 'brainfuck'],
34                                     default=
↳ 'python',
35                                     dot_
↳ cookiетemplate=dot_cookiетemplate,
36                                     to_get_
↳ property='language')
37
38     # prompt the user to fetch general template configurations
39     super().prompt_general_template_configuration(dot_cookiетemplate)
40
41     # switch case statement to prompt the user to fetch template specific
↳ configurations
42     switcher = {
43         'java': self.cli_java_options,
44         'brainfuck': self.cli_brainfuck_options
45     }
46     switcher.get(self.cli_struct.language)(dot_cookiетemplate)
47
48     self.cli_struct.is_github_repo, \
49     self.cli_struct.is_repo_private, \
50     self.cli_struct.is_github_orga, \
51     self.cli_struct.github_orga \
52     = prompt_github_repo(dot_cookiетemplate)
53
54     if self.cli_struct.is_github_orga:
55         self.cli_struct.github_username = self.cli_struct.github_orga
56
57     # create the chosen and configured template
58     super().create_template_without_subdomain(f'{self.TEMPLATES_CLI_PATH}')
59
60     # switch case statement to fetch the template version
61     switcher_version = {
62         'java': self.CLI_JAVA_TEMPLATE_VERSION,
63         'brainfuck': self.CLI_BRAINFUCK_TEMPLATE_VERSION
64     }
65     self.cli_struct.template_version, self.cli_struct.template_handle = switcher_
↳ version.get(
66         self.cli_struct.language.lower(), f'cli-{self.cli_struct.language.lower()}'
67
68     super().process_common_operations(path=Path(path).resolve(), domain='cli',
↳ language=self.cli_struct.language, dot_cookiетemplate=dot_cookiетemplate)
69

```

(continues on next page)

(continued from previous page)

```

70     def cli_python_options(self, dot_cookies: dict or None):
71         """ Prompts for cli-python specific options and saves them into the
72         ↪ CookieTemplateTemplateStruct """
73         self.cli_struct.command_line_interface = cookiequestionary_or_dot_
74         ↪ cookiequestionary(function='select',
75         ↪ question='Choose a command line library',
76         ↪ choices=['Click', 'Argparse', 'No command-line interface'],
77         ↪ default='Click',
78         ↪ dot_cookies=dot_cookies,
79         ↪ to_get_property='command_line_interface')
80         [...]
81     def cli_java_options(self, dot_cookies: dict or None) -> None:
82         """ Prompts for cli-java specific options and saves them into the
83         ↪ CookieTemplateTemplateStruct """
84         [...]
85     def cli_brainfuck_options(self):
86         """ Prompts for cli-brainfuck specific options and saves them into the
87         ↪ CookieTemplateTemplateStruct """
88         pass
    
```

4. If a new template were added we would also have to import our new Creator in create/create.py and add the new domain to the domain prompt and the switcher.

However, in this case we can simply skip this step, since cli is already included.

```

1  def choose_domain(domain: str):
2      """
3      Prompts the user for the template domain.
4      Creates the .cookiequestionary file.
5      Prompts the user whether or not to create a Github repository
6      :param domain: Template domain
7      """
8      if not domain:
9          domain = click.prompt('Choose between the following domains',
10                               type=click.Choice(['cli', 'gui', 'web', 'pub']))
11
12      switcher = {
13          'cli': CliCreator,
14          'web': WebCreator,
15          'gui': GuiCreator,
16          'pub': PubCreator
17      }
18
19      creator_obj = switcher.get(domain.lower())()
20      creator_obj.create_template()
    
```

5. Linting is up next! We need to ensure that our brainfuck template always adheres to the highest standards! Let's

edit `lint/domains/cli.py`.

We need to add a new class, which inherits from `TemplateLinter` and add our linting functions to it.

```

1 class CliBrainfuckLint(TemplateLinter, metaclass=GetLintingFunctionsMeta):
2     def __init__(self, path):
3         super().__init__(path)
4
5     def lint(self):
6         super().lint_project(self, self.methods)
7
8     def check_sync_section(self) -> bool:
9         """
10         Check the sync_files_blacklisted section containing every required file!
11         """
12         config_linter = ConfigLinter(f'{self.path}/cookiетemplate.cfg', self)
13         result = config_linter.check_section(section_items=config_linter.parser.items(
14             'sync_files_blacklisted'), section_name='sync_files_blacklisted',
15             main_linter=self, blacklisted_sync_files=[(
16             'changelog', 'CHANGELOG.rst')], -1],
17             error_code='cli-brainfuck-2', is_sublinter_
18             calling=True)
19         if result:
20             self.passed.append(('cli-brainfuck-2', 'All required sync blacklisted files_
21             are configured!'))
22         else:
23             self.failed.append(('cli-brainfuck-2', 'Blacklisted sync files section_
24             misses some required files!'))
25         return result
26
27     def brainfuck_files_exist(self) -> None:
28         """
29         Checks a given pipeline directory for required files.
30         Iterates through the templates's directory content and checkmarks files for_
31         presence.
32         Files that must be present::
33             'hello.bf',
34         Files that should be present::
35             '.github/workflows/build_brainfuck.yml',
36         Files that must not be present::
37             none
38         Files that should not be present::
39             none
40         """
41
42         # NB: Should all be files, not directories
43         # List of lists. Passes if any of the files in the sublist are found.
44         files_fail = [
45             ['hello.bf'],
46         ]
47         files_warn = [
48             [os.path.join('.github', 'workflows', 'build_brainfuck.yml')],
49         ]

```

(continues on next page)

(continued from previous page)

```

45     # List of strings. Fails / warns if any of the strings exist.
46     files_fail_ifexists = [
47
48     ]
49     files_warn_ifexists = [
50
51     ]
52
53     files_exist_linting(self, files_fail, files_fail_ifexists, files_warn, files_
    ↪warn_ifexists)

```

We need to ensure that our new linting function is found when linting is applied. Therefore, we turn our eyes to `lint/lint.py`, import our `CliBrainfuckLinter` and add it to the switcher.

```

1  from cookiетemplate.lint.domains.cli import CliBrainfuckLint
2
3  switcher = {
4      'cli-python': CliPythonLint,
5      'cli-java': CliJavaLint,
6      'cli-brainfuck': CliBrainfuckLint,
7      'web-website-python': WebWebsitePythonLint,
8      'gui-java': GuiJavaLint,
9      'pub-thesis-latex': PubLatexLint
10 }

```

Our shiny new `CliBrainfuckLinter` is now ready for action!

6. The only thing left to do now is to add a new Github Actions workflow for our template. Let's go one level up in the folder tree and create `.github/workflows/create_cli_brainfuck.yml`.

We want to ensure that if we change something in our template, that it still builds!

```

1  name: Create cli-brainfuck Template
2
3  on: [push]
4
5  jobs:
6      build:
7
8          runs-on: ubuntu-latest
9          strategy:
10             matrix:
11                 python: [3.8, 3.9]
12
13             steps:
14                 - uses: actions/checkout@v2
15                   name: Check out source-code repository
16
17                 - name: Setup Python
18                   uses: actions/setup-python@v2.2.2
19                   with:
20                       python-version: ${ matrix.python }
21
22                 - name: Install Poetry

```

(continues on next page)

(continued from previous page)

```

23     run: |
24         pip install poetry
25
26     - name: Build cookietemple
27       run: |
28         make install
29
30     - name: Create cli-brainfuck Template
31       run: |
32         echo -e "cli\nbrainfuck\nHomer\nhomer.simpson@hotmail.com\
↪nExplodingSpringfield\ndescription\nhomergithub\n" | poetry run cookietemple create
33
34     - name: Build Package
35       uses: fabasoad/setup-brainfuck-action@master
36       with:
37         version: 0.1.dev1
38     - name: Hello World
39       run: |
40         brainfucky --file ExplodingSpringfield/hello.bf

```

We were pleasantly surprised to see that someone already made a Github Action for brainfuck.

7. Finally, we add some documentation to `/docs/available_templates.rst` and explain the purpose, design and frameworks/libraries.

That's it! We should now be able to try out your new template using `cookiетemple create`. The template should be creatable, it should automatically lint after the creation and Github support should be enabled as well! If we run `cookiетemple list` Our new template should show up as well! I'm sure that you noticed that there's not actually a brainfuck template in cookietemple (yet!).

To quote our mighty Math professors: 'We'll leave this as an exercise to the reader.'

EXTERNAL PYTHON BASED PROJECTS

To use cookietemple in an external Python based project

```
import cookietemple
```


FAQ

cookiетemple is compound software and due to its complex nature many questions may arise. This section serves as a collection of frequently asked questions. If you do not find your answer here you may always [join our Discord channel](#) and ask for help. We are happy to include your question here afterwards.

16.1 I need help with cookiетemple. How can I get in contact with the developers?

You can [open an issue](#) or [join our Discord channel](#).

16.2 I am looking for a template for domain x and language y, but it does not exist yet!

We are always looking to add new templates to cookiетemple. Please [open an issue](#) or [join our Discord channel](#). Even better if you already have a draft for the template and/or could add it yourself!

TROUBLESHOOTING

All currently known issues can be found on our Github issue tracker. If there are any major known issues they will be listed here.

COMMUNITY

cookietemple is a huge community effort and can only be build with the combined expertise of people from all over the world.

No one knows all languages and ecosystems perfectly and we therefore want to invite everyone to join and contribute to cookietemple.

Please join our [Discord Channel](#) to discuss all things cookietemple and get help.

Please visit [Contributing](#) to learn how you can help and improve cookietemple! The easiest way is to spread the word.

18.1 Development Leads

- Lukas Heumos ([@zethson Github](#), [@Lukas Heumos Twitter](#))
- Philipp Ehmele ([@imipenem Github](#), [@Philipp Ehmele Twitter](#))

18.2 Core contributors

None yet. Why not be the first?

18.3 Contributors

- Filip Dutescu ([@filipdutescu Github](#)) (C++ template and more)
- Tobias Langes ([@adlanto Github](#)) (Windows support)

CONTRIBUTOR COVENANT CODE OF CONDUCT

19.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

19.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

19.3 Our Responsibilities

Maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

19.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

19.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by opening an issue. The project team will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

19.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at <https://www.contributor-covenant.org/version/1/4/code-of-conduct.html>

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`