# Flask

## Introduction

Flask is an API of Python that allows the developer to build up web-applications. It was developed by Armin Ronacher. Flask’s framework is more explicit than Django’s framework and is also easier to learn because it has less base code to implement in a simple web-Application. A Web-Application Framework or Web Framework is the collection of modules and libraries that helps the developer to write applications without writing the low-level codes such as protocols, thread management and so forth.2

Flask is therefore a micro web framework written in Python. It depends on the Jinja template engine and the Werkzeug WSGI toolkit. The “micro” in microframework means that Flask aims to keep the core simple but extensible. Flask won’t make many decisions for the developer, such as what database to use. Those decisions that it does make however, such as what templating engine to use, are easy to change. Virtually, everything else is up to the developer, so that Flask can be everything they need and nothing they don’t.

Interestingly, by default, Flask does not include a database abstraction layer, form validation or anything else where different libraries already exist that can handle that. Instead, Flask supports extensions to add such functionality to the application as if it was implemented in Flask itself. Numerous extensions provide database integration, form validation, upload handling, various open authentication technologies, and more. Flask may be “micro”, but it’s ready for production use on a variety of needs. Flask has many configuration values, with sensible defaults, and a few conventions when getting started. By convention, templates and static files are stored in subdirectories within the application’s Python source tree, with the names templates and static respectively.1

## Thread-Locals in Flask

One of the design decisions in Flask was that simple tasks should be simple; they should not take a lot of code and yet they should not limit you. Because of that, Flask has a few design choices that some people might find surprising or unorthodox. For example, Flask uses thread-local objects internally so that you don’t have to pass objects around from function to function within a request in order to stay threadsafe. This approach is convenient, but requires a valid request context for dependency injection or when attempting to reuse code which uses a value pegged to the request. The Flask project is honest about thread-locals, does not hide them, and calls out in the code and documentation where they are used. 1

## Security Concerns when using Flask

While writing a web application, the developer is probably allowing users to register and leave their data on his or her server. Indeed, such is the case in this thesis, where the user is required to create an account in order to use the application. In that regard, the users are entrusting the developer with their data. Even if it was just the developer submitting his or her data on the application, they still would want that data to be stored securely.

Unfortunately, there are many ways the security of a web application can be compromised. Flask protects against one of the most common security problems of modern web applications: cross-site scripting (XSS). Unless an insecure HTML is deliberately marked as secure, Flask and the underlying Jinja2 template engine have the developer covered. But there are many more ways to cause security problems.

The documentation does warn about aspects of web development that require attention to security. Some of these security concerns are far more complex than one might think, and everyone will occasionally underestimate the likelihood that a vulnerability will be exploited - until a clever attacker figures out a way to exploit their applications. Thinking also that your application is not important enough to attract an attacker is a grave mistake. Depending on the kind of attack, chances are that automated bots are probing for ways to fill the database with spam, links to malicious software, and the like.

Flask is no different from any other framework in that the developer must build with caution, watching for exploits when building to requirements. In that regard there has been a considerable effort to shield the users’ passwords, as well as guarantee that their inputs cannot, in any way abuse the system.1

### The passlib.hash module

The passlib.hash module contains all the password hash algorithms built into Passlib. While each hash has its own options and output format, they all inherit from the PasswordHash base interface. To achieve secure hashing however, one needs to be a bit careful. The module even warns the developer, that in order to achieve a secure hashing, he currently has four *good* choices:

* argon2
* bcrypt
* pbkdf2\_sha256 / pbkdf2\_sha512
* sha256\_crypt / sha512\_crypt

as those four hashes share some vital properties:

* Have no known vulnerabilities.
* Are based on documented and widely reviewed algorithms.
* Public-domain or BSD-licensed reference implementations are available.
* Have variable rounds for configuring flexible cpu cost on a per-hash basis.
* They have at least 96 bits of salt.
* Their basic algorithm has seen heavy scrutiny and use for at least 10 years (except for Argon2, born around 2013).
* Are in use across a number of OSes and/or a wide variety of applications.

Obviously, to properly hash the users’ passwords, one of the four algorithms has to be selected. Thankfully, this can come down to a series of questions.

1. Does the hash need to be natively supported by your operating system’s crypt() api, in order to allow inter-operation with third-party applications on the host?
   1. If yes, the right choice is either bcrypt for BSD variants, or sha512\_crypt for Linux; since these are natively supported.
   2. If no, continue…
2. Does your hosting provider allow you to install C extensions?
   1. If no, you probably want to use pbkdf2\_sha256, as this currently has the fastest pure-python backend.
   2. If they allow C extensions, continue…
3. Do you want to use the latest and greatest, and don’t mind increased memory usage when hashing?
   1. argon2 is a next-generation hashing algorithm, attempting to become the new standard. Its design has been being slightly tweaked since 2013, but will quite likely become the standard in the next few years. You’ll need to install the argon2\_cffi support library.
   2. If you want something secure, but more battle tested, continue…
4. The top choices left are bcrypt and pbkdf2\_sha256.

*Both have advantages, and their respective rough edges; though currently the balance is in favor of bcrypt (pbkdf2 can be cracked somewhat more efficiently).*

* 1. If choosing bcrypt, it is strongly recommend installing the bcrypt support library on non-BSD operating systems.
  2. If choosing pbkdf2, especially on python2 < 2.7.8 and python 3 < 3.4, it is advised to install fastpbk2 support library.

Considering what this thesis wants to achieve through hashing, neither the latest and greatest is required, nor any increased memory usage. What is simply required is a way to secure users’ passwords and doing so in the simplest (but definitely secure) way possible is ideal. Therefore the choice of pbkdf2\_sha256 was made.

#### Generic PBKDF2 Hashes – pbkdf2\_sha256

Passlib provides three custom hash schemes based on the PBKDF2 [1] algorithm which are compatible with the modular crypt format:

* pbkdf2\_sha1
* pbkdf2\_sha256
* pbkdf2\_sha512

Security-wise, PBKDF2 is currently one of the leading key derivation functions, and has no known security issues. Though the original PBKDF2 specification uses the SHA-1 message digest, it is not vulnerable to any of the known weaknesses of SHA-1, and can be safely used. However, for those still concerned, SHA-256 and SHA-512 versions are offered as well. PBKDF2-SHA512 is one of the four hashes Passlib recommends for new applications.3

### Concerning User Inputs

As mentioned before, a good practice for a developer is to always be vigilant for security threats, but also be proactive about them. Many times those security threats will target the database and try to either spam it with useless information or malicious links.

Thinking about the security risks of the current application, one can easily see the possibility of an ‘attack’ through user inputs throughout the application. As such, extensive measures should be taken to protect the database from undesired interactions and therefore, inputs should be monitored to guarantee they match the expected values, otherwise they should return the appropriate error. Upon inspecting the application at the later point, several examples of this effort will be presented.

# Bibliography

[1] Flask Documentation (https://flask.palletsprojects.com/)

[2] GeeksforGeeks definition (<https://www.geeksforgeeks.org/python-introduction-to-web-development-using-flask/>)

[3] Passlib.hash Documentations (https://passlib.readthedocs.io/en/stable/lib/passlib.ha-sh.html)