The Web2py Framework Documentation:

Introduction

web2py[**web2py**] is a free, open-source web framework for agile development of secure database-driven web applications; it is written in Python[**python**] and programmable in Python. web2py is a full-stack framework, meaning that it contains all the components you need to build fully functional web applications.

web2py is designed to guide a web developer to follow good software engineering practices, such as using the Model View Controller (MVC) pattern. web2py separates the data representation (the model) from the data presentation (the view) and also from the application logic and workflow (the controller). web2py provides libraries to help the developer design, implement, and test each of these three parts separately, and makes them work together.

web2py is built for security. This means that it automatically addresses many of the issues that can lead to security vulnerabilities, by following well established practices. For example, it validates all input (to prevent injections), escapes all output (to prevent cross-site scripting), renames uploaded files (to prevent directory traversal attacks). web2py leaves little choice to application developers in matters related to security.

Principles

Python programming typically follows these basic principles:

- Don't repeat yourself (DRY).
- There should be only one way of doing things.
- Explicit is better than implicit.

web2py fully embraces the first two principles by forcing the developer to use sound software engineering practices that discourage repetition of code. web2py guides the developer through

almost all the tasks common in web application development (creating and processing forms, managing sessions, cookies, errors, etc.).

web2py differs from other frameworks with regard to the third principle, which sometimes conflicts with the other two. In particular, web2py does not import user applications, but executes them in a predefined context. This context exposes the Python keywords, as well as the web2py keywords.

Web frameworks

At its most fundamental level, a web application consists of a set of programs (or functions) that are executed when the corresponding URL is visited. The output of the program is returned to the visitor and rendered by the browser.

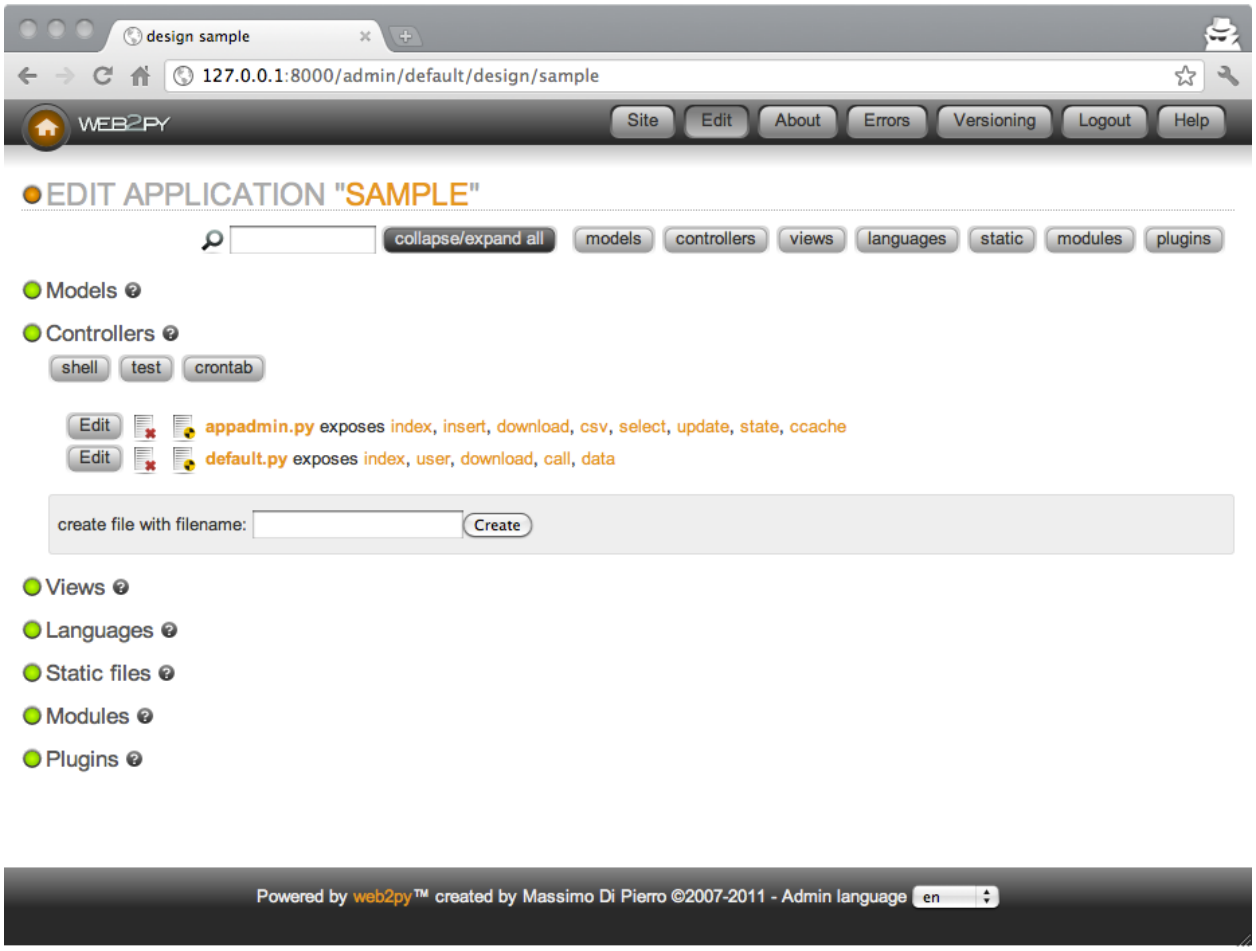
The purpose of web frameworks is to allow developers to build new apps quickly, easily and without mistakes. This is done by providing APIs and tools that reduce and simplify the amount of coding that is required.

The two classic approaches for developing web applications are:

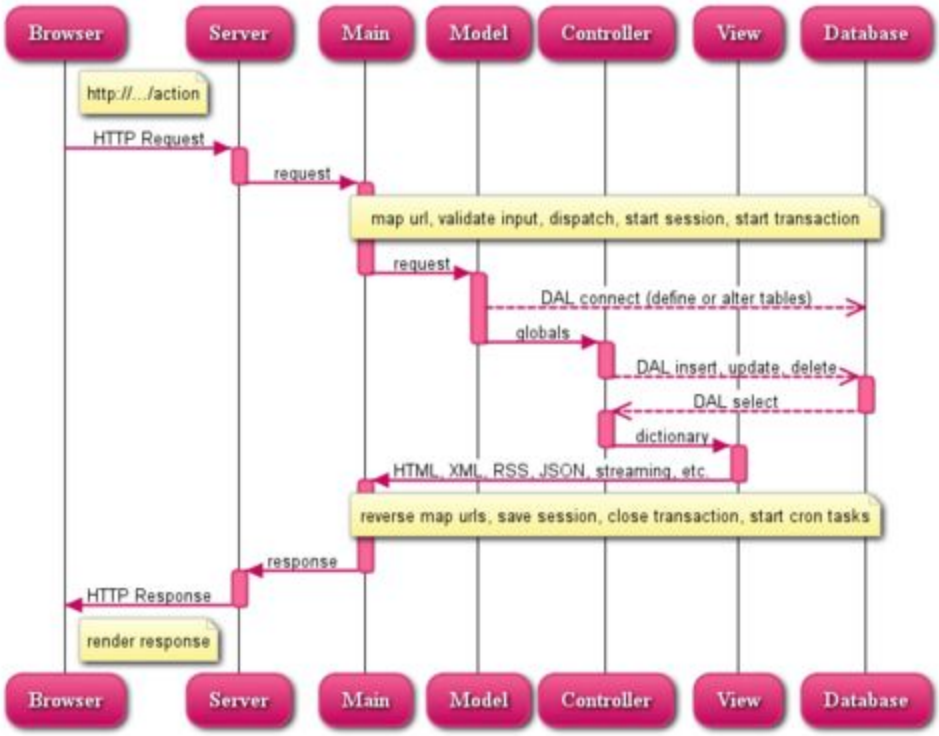
- Generating HTML[**html:w,html:o**] programmatically.
- Embedding code into HTML pages.

Model-View-Controller

web2py encourages the developer to separate data representation (the model), data presentation (the view) and the application workflow (the controller). Let's consider again the previous example and see how to build a web2py application around it. Here is an example of the web2py MVC edit interface:



The typical workflow of a request in web2py is described in the following diagram:



In the diagram:

- The Server can be the web2py built-in web server or a third-party server, such as Apache. The Server handles multi-threading.
- "main" is the main WSGI application. It performs all common tasks and wraps user applications. It deals with cookies, sessions, transactions, URL routing and reverse routing, and dispatching.

It can serve and stream static files if the web server is not doing it already.

- The Models, Views and Controller components make up the user application.
- Multiple applications can be hosted in the same web2py instance.
- The dashed arrows represent communication with the database engine(s). The database queries can be written in raw SQL (discouraged) or by using the web2py Database Abstraction Layer (recommended), so that web2py application code is not dependent on the specific database engine.
- The dispatcher maps the requested URL to a function call in the controller. The output of the function can be a string or a dictionary of symbols (a hash table). The data in the dictionary is rendered by a view. If the visitor requests an HTML page

(the default), the dictionary is rendered into an HTML page. If the visitor requests the same page in XML, web2py tries to find a view that can render the dictionary in XML. The developer can create views to render pages in any of the already supported protocols (HTML, XML, JSON, RSS, CSV, RTF) or in additional custom protocols.

- All calls are wrapped into a transaction, and any uncaught exception causes the transaction to be rolled back. If the request succeeds, the transaction is committed.
- web2py also handles sessions and session cookies automatically, and when a transaction is committed, the session is also stored, unless specified otherwise.
- It is possible to register recurrent tasks (via cron) to run at scheduled times and/or after the completion of certain actions. In this way it is possible to run long and compute-intensive tasks in the background without slowing down navigation.

Here is a minimal and complete MVC application, consisting of three files:

"db.py" is the model:

```
db = DAL('sqlite://storage.sqlite')
db.define_table('contact',
  Field('name'),
  Field('phone'))
```

It connects to the database (in this example a SQLite database stored in the `storage.sqlite` file) and defines a table called `contact`. If the table does not exist, web2py creates it and, transparently and in the background, generates SQL code in the appropriate SQL dialect for the specific database engine used. The developer can see the generated SQL but does not need to change the code if the database back-end, which defaults to SQLite, is replaced with MySQL, PostgreSQL, MSSQL, FireBird, Oracle, DB2, Informix, Interbase, Ingres, and the Google App Engine (both SQL and NoSQL).

Once a table is defined and created, web2py also generates a fully functional web-based database administration interface, called **appadmin**, to access the database and the tables.

"default.py" is the controller:

```
def contacts():
    grid=SQLFORM.grid(db.contact, user_signature=False)
    return locals()
```

In web2py, URLs are mapped to Python modules and function calls. In this case, the controller contains a single function (or "action") called contacts. An action may return a string (the returned web page) or a Python dictionary (a set of key:value pairs) or the set of local variables (as in this example). If the function returns a dictionary, it is passed to a view with the same name as the controller/function, which in turn renders the page. In this example, the function contacts generates a select/search/create/update/delete grid for table db.contact and returns the grid to the view.

"default/contacts.html" is the view:

```
{{extend 'layout.html'}}
<h1>Manage My Contacts</h1>
{{=grid}}
```

This view is called automatically by web2py after the associated controller function (action) is executed. The purpose of this view is to render the variables in the returned dictionary (in our case grid) into HTML. The view file is written in HTML, but it embeds Python code delimited by the special {{ and }} delimiters. This is quite different from the PHP code example, because the only code embedded into the HTML is "presentation layer" code. The "layout.html" file referenced at the top of the view is provided by web2py and constitutes the basic layout for all web2py applications. The layout file can easily be modified or replaced.

Why web2py?

web2py is one of many web application frameworks, but it has compelling and unique features. web2py was originally developed as a teaching tool, with the following primary motivations:

- Easy for users to learn server-side web development without compromising functionality. For this reason, web2py requires no installation and no configuration, has no dependencies (except for the source code distribution, which requires Python 2.5 and its standard library modules), and exposes most of its functionality via a Web browser interface.

- web2py has been stable from day one because it follows a top-down design; i.e., its API was designed before it was implemented. Even as new functionality has been added, web2py has never broken backwards compatibility, and it will not break compatibility when additional functionality is added in the future.

Architecture:

web2py is composed of the following components:

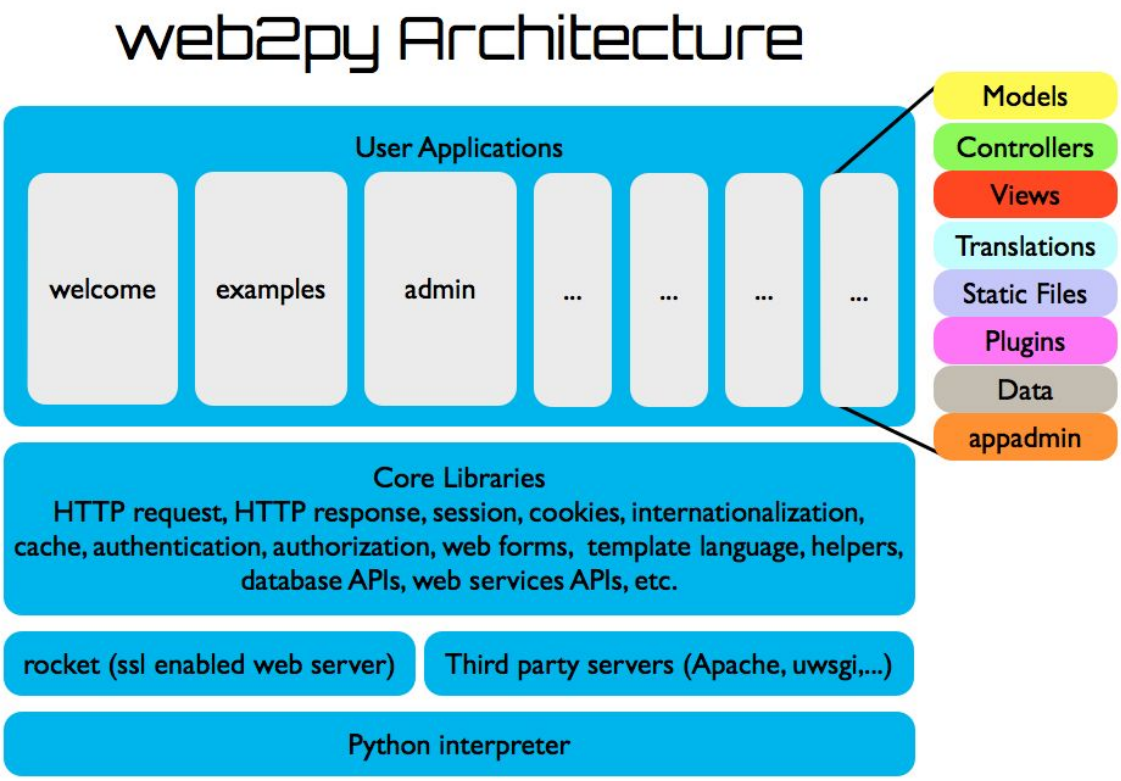
- **libraries**: provide core functionality of web2py and are accessible programmatically.
- **web server**: the Rocket WSGI web server.
- the **admin** application: used to create, design, and manage other web2py applications. **admin** provide a complete web-based Integrated Development Environment (IDE) for building web2py applications. It also includes other functionality, such as web-based testing and a web-based shell.
- the **examples** application: contains documentation and interactive examples. **examples** is a clone of the official web2py.com web site, and includes epydoc documentation.
- the **welcome** application: the basic scaffolding template for any other application. By default it includes a pure CSS cascading menu and user authentication (discussed in Chapter 9).

web2py is distributed in source code, and in binary form for Microsoft Windows and for Mac OS X.

The source code distribution can be used in any platform where Python runs and includes the above-mentioned components. To run the source code, you need Python 2.5 pre-installed on the system. You also need one of the supported database engines installed. For testing and light-demand applications, you can use the SQLite database, included with Python 2.5.

The binary versions of web2py (for Windows and Mac OS X) include a Python 2.5 interpreter and the SQLite database. Technically, these two are not components of web2py. Including them in the binary distributions enables you to run web2py out of the box.

The following image depicts the overall web2py structure:



web2py™ Download

For Python 3.7	For Python 2.7	For Testers	For Developers
Windows binaries	Windows binaries	Windows binaries	Git Repository
Mac binaries	Mac binaries	Mac binaries	Manual
Source Code	Source Code	Source Code	Source code docs
		Change Log	Report a Bug

The source code version works on Windows and most Unix systems, including **Linux**, **BSD** and **Mac** . It requires Python 3.5+ (recommended for new projects) or Python 2.7+ (stable, for use with legacy apps) already installed on your system.

There are also binary packages for Windows and MacOs. They include the Python interpreter version 3.7.4 or 2.7.16, so you do not need to have it pre-installed.

Installation:

Instructions

With the binary packages, after download, just unzip it and then click on web2py.exe (Windows) or web2py (MacOs).

Note that on recent MacOs versions (10.12+) you could face problems in running the binary App program, due to the last changes to the security settings. In this case, press the 'control' key + click on downloaded file and then 'open' it (confirm the warnings). Finally move the program in Applications and run it from there.

If you prefer to run it from source with your own Python interpreter already installed, type:

```
>python
  web2py.py
```

or for more info type:

```
>python web2py.py
  -h
```

Caveats

After installation, every time you run it, web2py asks you to choose a password. This password is your administrative password. If the password is left blank, the administrative interface is disabled. The administrative interface /admin/default/index is only accessible via localhost and always requires a password.

Any url /a/b/c maps into a call to application a, controller b.py and function c in that controller.

You are strongly advised to also use Apache with mod_proxy or mod_wsgi to access applications in the framework. This allows better security and concurrency.

Example:

A web2py application can be as simple as a single file (controllers/default.py) containing:

When http://localhost:8000/app/default/index is visited the function is called and it displays the message "Hello World".
Here is a more complex complete application that lets the visitor upload images into a database:

In Model

```
1.    DAL 'sqlite://storage.db'
2.                        'image'
3.  Field 'name'              True
4.  Field 'file' 'upload'
```

In Controller

```
1.def
2.            SQLFORM
3.  if
4.    response          'image uploaded'
5.  return
```

In View

```
1.            'layout.html'
2.
3.
```

Uploaded images are safely renamed to avoid directory traversal vulnerabilities, stored on the filesystem (or database) and a corresponding entry is inserted in the database, linking the file. A built-in mechanism prevents involuntary double form submission. All DB IO is transaction safe by default. Any exception in the code causes the transaction to rollback.

Advantages:

- **Web2py** is a potential framework when compared to Django and Flask in terms of speed and simplicity of the development. ...
- **Web2py** can run python compiled code as an optimization to lower the running time and to allow you to distribute your code in a compiled fashion.

Disadvantages:

- Web2py supports doctests, however it does not support unit testing. Now doctests are not the optimal choice because of their limited scope.
- There is no differentiation between production and development mode. In case an exception occurred, ticket is generated all the times and you will have to navigate to the ticket to check the error. This might be helpful in case of production server but will be difficult in development environment as developers really need to see the error instantly rather than checking the ticket number.
- Web2py has a good database abstraction layer (DAL) that allows you to abstract many types of database engines but it lacks powerful ORM. In case you are dealing with relatively large model, your code will get scattered by all nested definitions and attributes which makes things complicated.

- We cannot use standard python development tools without modifications as web2py has really poor IDE support.

Python web frameworks are high-level tools used for the development of web applications, APIs and more. Frameworks facilitate easy and speedy development of dynamic and robust web resources. While many frameworks for Python exist, this article focuses on Django and web2py and their differences.

Framework overview

Django is notably one of the most popular Python web frameworks in existence. It is a high-level open source full-stack framework with a focus on rapid development and clean design, and it allows you to build dynamic, well-structured applications. It is also based on the Don't Repeat Yourself (DRY) principle. This means exactly what it sounds like. The idea is that there is no point in writing the same code over and over, so the framework provides a lot of features to prevent this. This makes Django one of the fastest frameworks there is. Django is well known for its templating features, Object Relational Mapping (ORM) tool, Class-Based Views, Admin, Routing and REST frameworks. It includes a lightweight standalone web server for development and testing.

Web2py is a scalable, open source full-stack framework for Python which comes with its own web-based IDE. Web2py has no requirements for installation and configuration, has readability features for multiple protocols, and can run on Windows, Mac, Linux/Unix, Google App Engine, Amazon EC2, and any web hosting that supports Python 2.5+. Web2py is also backwards-compatible. Its key features include Role-Based Access Control (RBAC), Database Abstraction Layer (DAL) and support for internationalization. It's worth noting that the design of web2py was inspired by Django.

Web2py's mantra is "simple is better than complex" and makes it easy to write simple, clean code, while Django values "explicit over implicit" and gives developers more control over the design/process.

WEB2PY vs DJANGO:

Database and server access, config.

- Django:Django can be run together with Apache, NGINX using WSGI, Gunicorn, or Cherokee. It also has the ability to use other WSGI-compliant web servers such as Bjoern. Django's ORM is one of its key features. It's more efficient for larger models than web2py's DAL.
- Django uses the Object-Relational Mapper to map objects to database tables. The framework's main databases are MySQL, PostgreSQL, SQLite, and Oracle. Django, however, does have a fork called django-nonrel, which supports NoSQL databases such as MongoDB.
- If Django-jython is installed, Django may be run with Jython on any Java EE application server, such as Tomcat or Jetty.
- Web2py: Web2py services requests to Apache, Lighttpd, Cherokee, NGINX and Hiawatha with its built-in Rocket Server. It can also service requests with pretty much any other web server out there using FastCGI, CGI, WSGI, mod_proxy or mod_python.
- Web2py comes with an API called the Database Abstraction Layer that maps Python objects into database objects such as queries and tables. The DAL allows you to specify a dialect for the database back-end, and then generates the SQL in real time. This means you don't have to write SQL at all. For detailed instructions on implementing databases using the DAL, read the web2py DAL documentation.

- Web2py also supports SQLite, MSSQL, MySQL, and PostgreSQL for Windows. For Mac, however, it only supports SQLite. To use other database back-ends, web2py allows users to install the appropriate drive for the back-end in question. This means that popular databases such as FireBird, MongoDB, etc. can be used with web2py.

Template:

- Django: While Django does have its own Django Template language, the learning curve is pretty gentle, and it's completely optional. Django also has template inheritance features. This means that every web app has a base template which can then be inherited by other pages. Template inheritance ensures that you maintain only one copy of your markup in the future, as opposed to multiple copies. Apart from being used to minimize repetition of HTML to structure pages, they can also be used to minimize code in application views. Django also allows you to create your own template tags.
- Web2py: In contrast, web2py has no special template language, and all controls and web templates are written in pure Python. They're also very simple to understand. Once you have the basic idea of how to write templates as views, you can start writing code.

Libraries/tutorials and the learning curve

- Django: Django has been around since July 2005, so there are several tutorials, libraries, and over 2500 packages etc. available to you. Django also has extensive documentation that can be accessed here. For a comprehensive list of reusable apps, tools, etc., visit https://djangopackages.org. Django has a slightly steep learning curve. Because its entire system is based on inheritance, it can be quite complex for people who are new at Python or not very familiar with object-oriented programming in general.
- Web2py: Web2py was developed in 2008. While the framework has a very helpful and active society which provides free plugins, applications and a lot of tutorials, it still has significantly less support than Django. Yet this is mitigated by the fact that web2py's overall learning curve is much easier in

comparison to Django. Web2py was specifically designed as a learning tool and is astonishingly easy to set up. All that is required is that you download the web2py distribution, start up its built-in Rocket web server, and get to coding.

What should you use, and when?

The philosophies of both frameworks are deeply rooted in Python, but there are some advantages to using one over the other in certain scenarios. If you're a beginner programmer, or a newbie at web dev, web2py is definitely for you. However, if you're fluent in Python and you need to meet a close deadline, Django would be the better option. Also, for popularity reasons, if you're a freelancer looking to land web development jobs, learning to use Django would be more suited to your needs as there are more jobs available for it.

s