

---

# **jetfactory Documentation**

**Robert Wikman <[rbw@vault13.org](mailto:rbw@vault13.org)>**

**Mar 10, 2019**



---

## Contents

---

<b>1</b>	<b>Compatibility</b>	<b>3</b>
<b>2</b>	<b>Installing</b>	<b>5</b>
<b>3</b>	<b>License</b>	<b>7</b>
<b>4</b>	<b>Environment</b>	<b>9</b>
<b>5</b>	<b>Instance</b>	<b>11</b>
<b>6</b>	<b>File</b>	<b>13</b>
<b>7</b>	<b>Application</b>	<b>15</b>
<b>8</b>	<b>Manager</b>	<b>17</b>
<b>9</b>	<b>Usage</b>	<b>19</b>
<b>10</b>	<b>Controller</b>	<b>21</b>
10.1	Routing . . . . .	21
10.2	Transformation . . . . .	22
10.3	Schemas . . . . .	22
<b>11</b>	<b>Services</b>	<b>25</b>
11.1	BaseService . . . . .	25
11.2	DatabaseService . . . . .	25
11.3	HttpClientService . . . . .	26



Jetfactory is a Python framework for creating structured, portable and high-performance Web APIs. It's built on top of Sanic and uses the blazing fast uvloop implementation of the asyncio event loop.



# CHAPTER 1

---

## Compatibility

---

Python 3.6+





## CHAPTER 2

---

### Installing

---

```
$ pip install jetfactory
```



## CHAPTER 3

---

### License

---

#### BSD 2-Clause License

Copyright (c) 2019, Robert Wikman <[rbw@vault13.org](mailto:rbw@vault13.org)> All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.



## CHAPTER 4

---

### Environment

---



## CHAPTER 5

---

Instance

---





## CHAPTER 6

---

File

---



# CHAPTER 7

---

## Application

---

At a minimum, the `jetfactory.Application` needs to be created with at least one *Jetpack*. Additional parameters can be provided to further customize the instance; see the docs below for more info.

### API

---

**Note:** The `jetfactory.Application` and `jetfactory.Application.run()` can be configured using the environment and files as well.

*Read more about this in the Configuration section.*

---

### Example

```
import jetfactory
import jet_apispec
import jet_guestbook

# Create application
app = jetfactory.Application(
    path='/api',
    packages=[
        ('/guestbook', jet_guestbook),
        ('/packages', jet_apispec)
    ]
)

# Start server
app.run(host='192.168.0.1')
```



## CHAPTER 8

---

Manager

---



## CHAPTER 9

---

### Usage

---

Jetpacks are used for grouping, labeling and making components ready for registration with an Application.

#### API

---

**Important:** The Jetpack's `__init__.py` file must have a `__version__` variable set to be successfully registered with an Application.

---

#### Example

```
from jetfactory import Jetpack
from jet_guestbook import service
from .service import VisitService, VisitorService
from .model import VisitModel, VisitorModel
from .controller import Controller

__version__ = '0.1.0'

export = Jetpack(
    controller=Controller,
    services=[VisitService, VisitorService],
    models=[VisitModel, VisitorModel],
    name='guestbook',
    description='Example guestbook package'
)
```





# CHAPTER 10

---

## Controller

---

The Jetpack *Controller* class inherits from `ControllerBase` and is registered with the `Application` upon server start.

### API

#### Example

```
from jetfactory.controller import ControllerBase

class Controller(ControllerBase):
    async def on_ready(self):
        self.log.debug(f'Controller ready at path: {self.pkg.path}')

    async def on_request(self, request):
        self.log.debug(f'Request received: {request}')
```

---

**Note:** Having a *Controller* without *Route handlers* is not of much use; continue reading to see how *handlers* can be added.

---

## 10.1 Routing

Routing is implemented using one or more *handlers* decorated with a `@route`. Used without the `@input_load` decorator, the entire request object is passed to the handler.

### API

#### Example

```
from sanic.response import HTTPResponse
from jetfactory.controller import ControllerBase, route
```

(continues on next page)

(continued from previous page)

```
class Controller(ControllerBase):
    async def on_request(self, request):
        self.log.debug(f'Request received: {request}')

    @route('/<name>', 'GET'):
    async def greet(self, request, name):
        return HTTPResponse({'msg': f'hello {name} from {request.ip}'})
```

## 10.2 Transformation

Request and response transformation is performed when a request reaches `@input_load`, and upon handler return in `@output_dump`. These two decorators provides a declarative way of defining what comes in and what goes out of a route handler.

### API

#### Example of request/response transformation

```
from jetfactory.controller import ControllerBase, route
from jetfactory.schema import ParamsSchema
from .visit import svc_visit
from .visit.schemas import Visit

class Controller(ControllerBase):
    async def on_request(self, request):
        self.log.debug(f'Request received: {request}')

    @route('/', 'GET')
    @input_load(query=ParamsSchema) # Transform and validate the query string
    @output_dump(Visit, many=True) # Dump many `Visit`s
    async def visits_get(self, query):
        # Call the service layer and dump the result as a JSON string
        return await svc_visit.get_many(**query)

    @route('/<visit_id>', 'PUT') # Perform an update operation
    @input_load(body=Visit) # Transform and validate the JSON payload
    @output_dump(Visit) # Dump one `Visit`
    async def visit_update(self, remote_addr, body, visit_id):
        # Call the service layer and dump the result as a JSON string
        return await svc_visit.visit_update(remote_addr, visit_id, body)
```

## 10.3 Schemas

Schemas are used in transformation decorators to perform object serialization and generating HTTP API documentation.

---

**Note:** Not familiar with Marshmallow schemas? Check out the [Marshmallow API docs](#) for info.

---

### Example

```
from jetfactory.schema import fields, Schema

class Visit(Schema):
    id = fields.Integer()
    visited_on = fields.String(attribute='created_on')
    message = fields.String()
    name = fields.String()

    class Meta:
        dump_only = ['id', 'visited_on']
        load_only = ['visit_id', 'visitor_id']

class VisitNew(Schema):
    message = fields.String(required=True)
    name = fields.String(required=True)
```



# CHAPTER 11

---

## Services

---

Jetfactory provides a set of built-in service classes, or Mixins if you will - used to extend a Package's service layer with extra features such as database and HTTP access.

---

**Note:** Create a PR or Issue if you want a Service Layer component added or updated.

---

### 11.1 BaseService

This Service implements the singleton pattern and is directly or indirectly used by all types of Jetfactory services.

API

### 11.2 DatabaseService

The built-in `DatabaseService` inherits from `BaseService` and provides an interface for interacting with MySQL and PostgreSQL databases using the [peewee-async manager](#).

#### 11.2.1 Example

```
from jetfactory.service import DatabaseService
from jetfactory.exceptions import JetfactoryException

from jet_guestbook.model import VisitModel

class VisitService(DatabaseService):
    __model__ = VisitModel

    async def get_authored(self, visit_id, remote_addr):
```

(continues on next page)

(continued from previous page)

```
visit = await self.get_by_pk(visit_id)
if visit.visitor.ip_addr != remote_addr:
    raise JetfactoryException('Not allowed from your IP', 403)

return visit

async def visit_count(self, ip_addr):
    return await self.count(VisitModel.visitor.ip_addr == ip_addr)
```

## 11.2.2 API

---

**Important:** The `__model__` class attribute must be set for Services implementing the *DatabaseService*.

---

## 11.2.3 Models

Models are implemented using `Peewee.Model`.

### Example

```
from datetime import datetime
from peewee import Model, ForeignKeyField, CharField, DateTimeField
from .visitor import VisitorModel

class VisitModel(Model):
    class Meta:
        table_name = 'visit'

    created_on = DateTimeField(default=datetime.now)
    message = CharField(null=False)
    visitor = ForeignKeyField(VisitorModel)

    @classmethod
    def extended(cls, *fields):
        return cls.select(VisitModel, VisitorModel, *fields).join(VisitorModel)
```

## 11.3 HttpClientService

The built-in `HttpClientService` provides an interface for interacting with HTTP servers.

### 11.3.1 Example

```
from jetfactory.service import HttpClientService, DatabaseService

from .model import EntryModel
```

(continues on next page)

(continued from previous page)

```
class EntryService(HttpClientService, DatabaseService):
    __model__ = EntryModel

    def __init__(self):
        self.backup_url = 'https://192.168.1.10'

    async def entry_add(self, entry_new):
        entry = await self.create(entry_new)

        self.log.info(f'sending a copy to {self.backup_url}')
        await self.http_post(self.backup_url, entry_new)

    return entry
```

### 11.3.2 API