Fusio Documentation

Release 1.0

Christoph Kappestein

Jul 24, 2017

Contents

Fusio is an open source API management platform which helps to build and manage RESTful APIs. The documentation covers the installation process, how to manage Fusio through the backend, how to use Fusio with 3rd party consumer applications and how to build reusable adapter components.

CHAPTER 1

TOC

Overview

About

Fusio is an open source API management platform which helps to build and manage RESTful APIs. We think that there is a huge potential in the API economy. Whether you need an API to expose your business functionality, build micro services or to develop One-Page web applications or Mobile-Apps. Because of this we think that Fusio is a great tool to simplify building such APIs. More information on http://www.fusio-project.org/

Features

Fusio covers all important aspects of the API lifecycle so you can concentrate on building the actual business logic of your API.

- **Versioning**

  It is possible to define different versions of your endpoint. A concrete version can be requested through the `Accept` header i.e. `application/vnd.acme.v1+json`

- **Documentation**

  Fusio generates automatically a documentation of the API endpoints based on the provided schema definitions.

- **Validation**

  Fusio uses the standard JSONSchema to validate incoming request data.

- **Authorization**

  Fusio uses OAuth2 for API authorization. Each app can be limited to scopes to request only specific endpoints of the API.

- **Analytics**

Fusio monitors all API activities and shows them on a dashboard so you always know what is happening with your API.

- **Rate limiting**

  It is possible to limit the requests to a specific threshold.

- **Specifications**

  Fusio generates different specification formats for the defined API endpoints i.e. OAI (Swagger), RAML.

- **User management**

  Fusio provides an API where new users can login or register a new account through GitHub, Google, Facebook or through normal email registration.

Basically with Fusio you only have to define the schema (request/response) of your API endpoints and implement the business logic in a simple PHP file. All other aspects are covered by Fusio.

Development

If you develop an API with Fusio you need to define a .fusio.yml deploy file which specifies the available routes and actions for the system. If a request schema is available for a method the input gets validated according to the schema. A deploy file looks like:

```
routes:
  "/todo":
    version: 1
    methods:
      GET:
        public: true
        response: Todo-Collection
        action: "${dir.src}/Todo/collection.php"
      POST:
        public: false
        request: Todo
        response: Todo-Message
        action: "${dir.src}/Todo/insert.php"
  "/todo/:todo_id":
    version: 1
    methods:
      GET:
        public: true
        response: Todo
        action: "${dir.src}/Todo/row.php"
      DELETE:
        public: false
        response: Todo-Message
        action: "${dir.src}/Todo/delete.php"
schema:
  Todo: !include resources/schema/todo/entity.json
  Todo-Collection: !include resources/schema/todo/collection.json
  Todo-Message: !include resources/schema/todo/message.json
connection:
  Default-Connection:
    class: Fusio\Adapter\Sql\Connection\SqlAdvanced
    config:
      url: "sqlite:///${dir.cache}/todo-app.db"
migration:
```

```
  Default-Connection:
    - resources/sql/v1_schema.sql
```

This file can be deploy with the following command:

```
php bin/fusio deploy
```

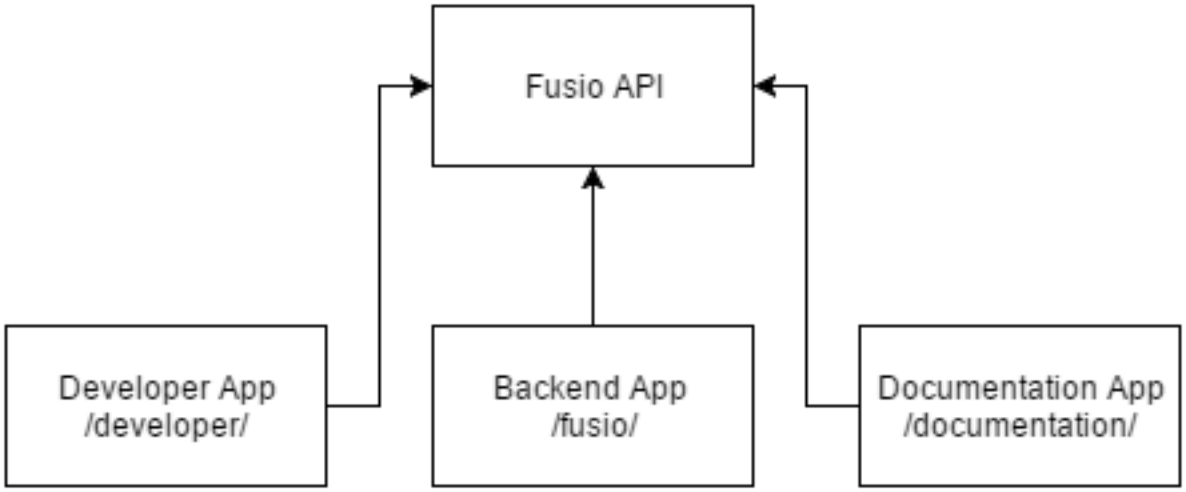
The action of each route contains the file which handles the business logic. By default we use the PhpFile engine which uses a simple PHP file but you can also set another engine i.e. PhpClass or V8 to use either an actual php class or javascript code. More information in the src/ folder. In the following an example action to build an API response from a database:

```
<?php
/**
 * @var \Fusio\Engine\ConnectorInterface $connector
 * @var \Fusio\Engine\RequestInterface $request
 * @var \Fusio\Engine\Response\FactoryInterface $response
 * @var \Fusio\Engine\ProcessorInterface $processor
 * @var \Psr\Log\LoggerInterface $logger
 * @var \Psr\SimpleCache\CacheInterface $cache
 */

/** @var \Doctrine\DBAL\Connection $connection */
$connection = $connector->getConnection('Default-Connection');

$count   = $connection->fetchColumn('SELECT COUNT(*) FROM app_todo');
$entries = $connection->fetchAll('SELECT * FROM app_todo WHERE status = 1 ORDER BY␣
↪insertDate DESC LIMIT 16');

return $response->build(200, [], [
    'totalResults' => $count,
    'entry' => $entries,
]);
```

In the code we get the Default-Connection which we have defined previously in our `.fusio.yml` deploy file. In this case the connection returns a `\Doctrine\DBAL\Connection` instance but we have already many adapters to connect to different services. Then we simply fire some queries and return the response.

Backend

Fusio provides several apps which work with the internal backend API. These apps can be used to manage and work with the API. This section gives a high level overview what the Fusio system provides and how the application is structured. Lets take a look at the components which are provided by Fusio:



API

If you install a Fusio system it setups the default API with that it is possible to manage the complete system. Because of that Fusio has some reserved paths which are needed by the system.

- `/backend`

  Endpoints for configuring the system

- `/consumer`

  Endpoints for the consumer i.e. register new accounts or create new apps

- `/doc`

  Endpoints for the documentation

- `/authorization`

  Endpoints for the consumer to get i.e. information about the user itself and to revoke an obtained access token

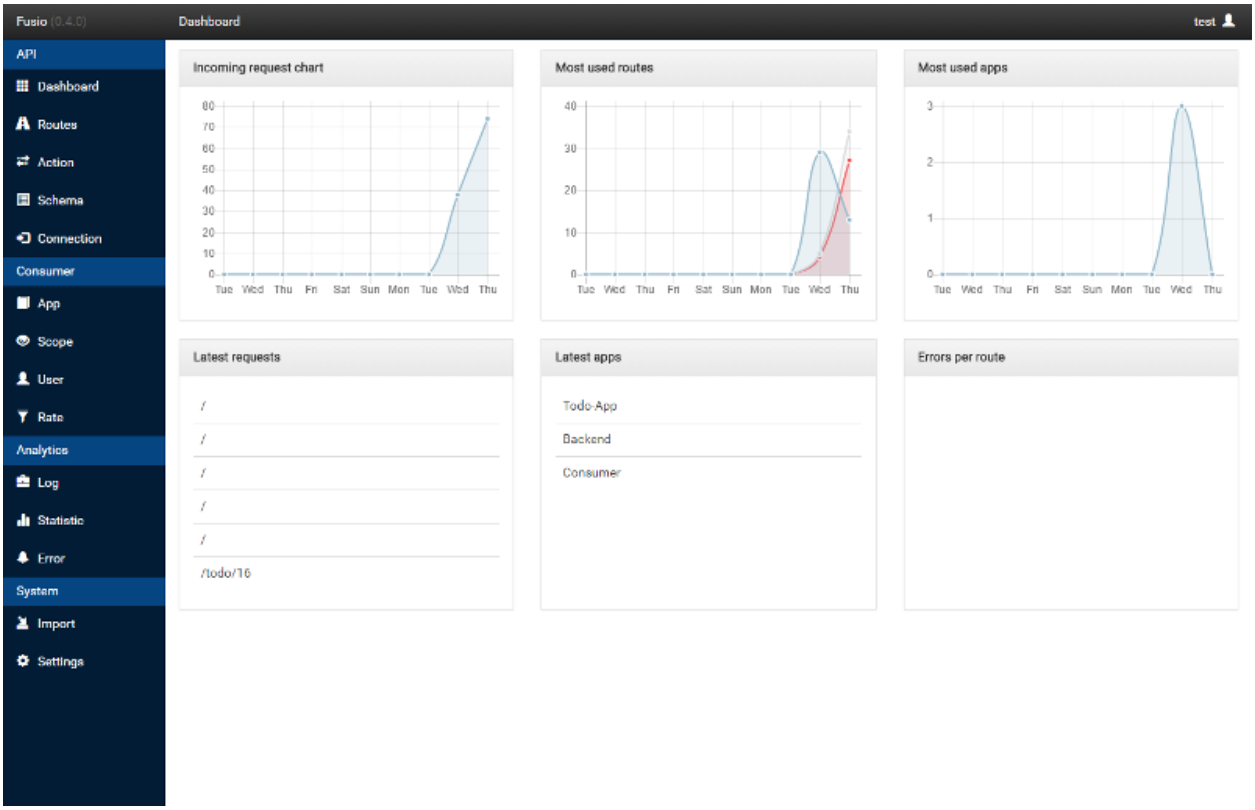
- `/export`

  Endpoints to export the documentation into other formats i.e. swagger

Apps

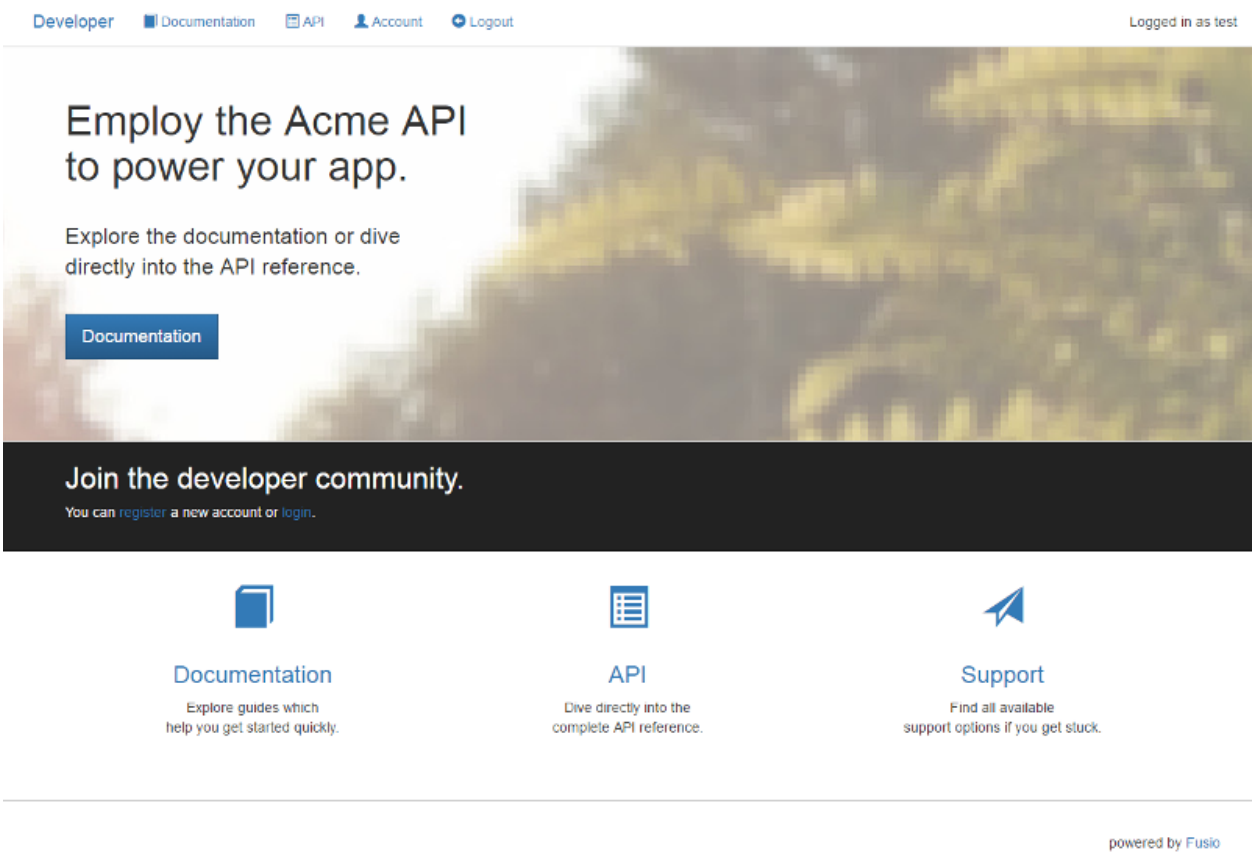
All following apps are working with the API. Because of that it is also really easy to integrate Fusio into an existing system since you can call the endpoints from your application.

Backend



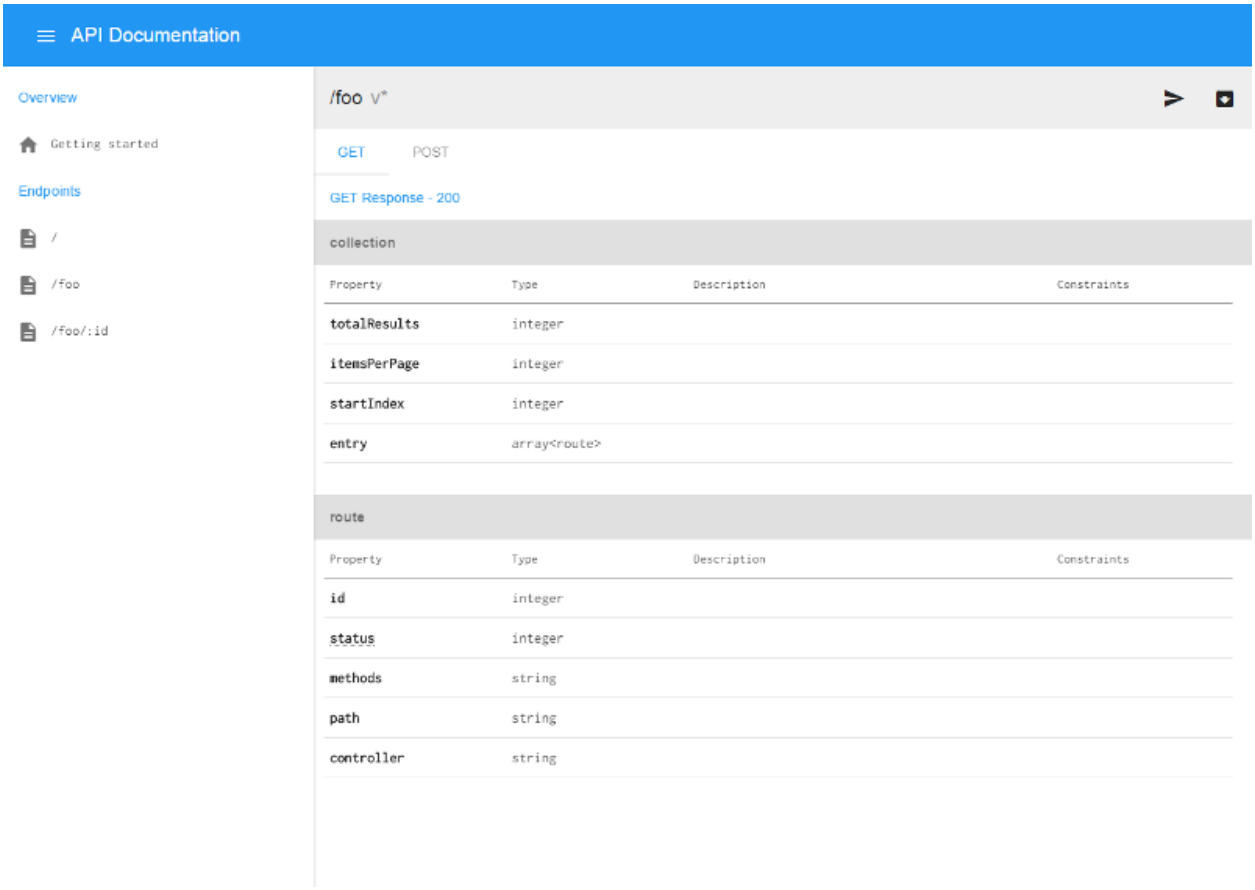
The backend app is the app where the administrator can configure the system. The app is located at `/fusio/`.

Developer



The developer app is designed to quickly setup an API programm where new developers can register and create/manage their apps. The app is located at `/developer/`.

Documentation



The documentation app simply provides an overview of all available endpoints. It is possible to export the API definition into other schema formats like i.e. Swagger. The app is located at `/documentation/`.

Installation

It is possible to install Fusio either through composer or install it manually. Place the project into the www directory of the web server.

Composer

```
composer create-project fusio/fusio
```

Download

https://github.com/apioo/fusio/releases

Configuration

- **Adjust the configuration file**

Open the file `configuration.php` in the Fusio directory and change the key `psx_url` to the domain pointing to the public folder. Also insert the database credentials to the `psx_connection` keys.

- **Execute the installation command**

  The installation script inserts the Fusio database schema into the provided database. It can be executed with the following command `php bin/fusio install`.

- **Create administrator user**

  After the installation is complete you have to create a new administrator account. Therefor you can use the following command `php bin/fusio adduser`. Choose as account type "Administrator".

You can verify the installation by visiting the `psx_url` with a browser. You should see a API response that the installation was successful. The backend is available at `/fusio/`.

Docker

Alternatively it is also possible to setup a Fusio system through docker. This has the advantage that you automatically get a complete running Fusio system without configuration. This is especially great for testing and evaluation. To setup the container you have to checkout the repository and run the following command:

```
docker-compose up -d
```

This builds the Fusio system with a predefined backend account. The credentials are taken from the env variables `FUSIO_BACKEND_USER`, `FUSIO_BACKEND_EMAIL` and `FUSIO_BACKEND_PW` in the docker-compose.yml. If you are planing to run the container on the internet you must change these credentials.

Configure web server

It is recommended to setup a virtual host in your `sites-available` folder which points to the public folder of Fusio. After this you also have to change the configuration of the url i.e.:

```
'psx_url' => 'http://api.acme.com',
```

Apache

```
<VirtualHost *:80>
    ServerName api.acme.com
    DocumentRoot /var/www/fusio/public/
    ErrorLog /var/log/apache2/fusio-error.log
    CustomLog /var/log/apache2/fusio-access.log combined
</VirtualHost>
```

You should enable the module `mod_rewrite` so that the .htaccess file in the public folder is used. The htaccess contains an important rule which redirects the `Authorization` header to Fusio which is otherwise removed. If the .htaccess file does not work please check whether the `AllowOverride` directive is set correctly i.e. to `All`.

Javascript V8

Fusio provides an adapter which lets you write the endpoint logic in simple javascript. To use this adapter you need to install the `php-v8` extension. Installation instructions are available at the php-v8 repository

Apps

There are three javascript apps which can connect to the Fusio backend API. The backend, developer and documentation app. By default they try to guess the url of the API endpoint. If an app is not working properly the problem is probably that the javascript app can not correctly determine the API endpoint url. In this case you have to adjust the url in the following files:

- `/public/fusio/index.htm`
- `/public/developer/index.html`
- `/public/documentation/index.html`

These apps are of course optional. If you dont want to use them you could also simply delete the folder.

Backend

At the endpoint `fusio/` you can login to the backend app. You should be able to login with the username (which you have entered for the `adduser` command) and the password which you have used. The following list covers the most login errors in case you are not able to login at the backend:

- **The javascript Backend-App uses the wrong API endpoint**

  This can be tested with the browser developer console. If you login at the backend with no credentials the app should make an request to the `/backend/token` endpoint which should return a JSON response i.e.:

  ```
  { "error": "invalid_request", "error_description": "Credentials not available" }
  ```

  If this is the case your app is correctly configured. If this is not the case you need to adjust the endpoint url at `/public/fusio/index.htm` i.e.:

  ```
  var fusioUrl = "http://localhost:8080/fusio/public/index.php/";
  ```

- **Apache module mod_rewrite is not activated**

  In case you use Apache as web server you must activate the module `mod_rewrite` so that the `public/.htaccess` file is used. Besides clean urls it contains an important rule which tells Apache to redirect the `Authorization` header to Fusio otherwise Apache will remove the header and Fusio can not authenticate the user

- **Fusio API returns an error**

  In this case Fusio can probably not write to the `cache/` folder. To fix the problem you have to change the folder permissions so that the user of the web server can write to the folder. If there is another error message it is maybe a bug. Please report the issue to GitHub.

Updating

There are two parts of Fusio which you can update. The backend system and the backend app. The backend app is the AngularJS application which connects to the backend api and where you configure the system. The backend system contains the actual backend code providing the backend API and the API which you create with the system.

Backend system

Fusio makes heavy use of composer. Because of that you can easily upgrade a Fusio system with the following composer command.

```
composer update fusio/impl
```

This has also the advantage that the version constraints of installed adapters are checked and in case something is incompatible composer will throw an error. It is also possible to simply replace the vendor folder with the folder from the new release. In either case you have to run the following command after you have updated the vendor folder:

```
php bin/fusio install
```

This gives Fusio the chance to adjust the database schema in case something has changed with a new release.

Backend app

To update the backend app simply replace the javascript and css files from the new release:

- `public/fusio/`

Get started

Build an API endpoint

Fusio provides a demo todo API which is ready for deployment. Take a look at the `.fusio.yml` file which contains the deployment configuration. The file contains several keys:

- **routes**

  Describes for each route the available request methods, whether the endpoint is public or private, the available request/response schema and also the action which should be executed:

```
routes:
  "/todo":
    version: 1
    methods:
      GET:
        public: true
        response: Todo-Collection
        action: "${dir.src}/Todo/collection.php"
      POST:
        public: false
        request: Todo
        response: Todo-Message
        action: "${dir.src}/Todo/insert.php"
  "/todo/:todo_id":
    version: 1
    methods:
      GET:
        public: true
        response: Todo
        action: "${dir.src}/Todo/row.php"
      DELETE:
        public: false
        response: Todo-Message
        action: "${dir.src}/Todo/delete.php"
```

- **schema**

Contains the available request and response schema in the JSON-Schema format:

```
schema:
  Todo: !include resources/schema/todo/entity.json
  Todo-Collection: !include resources/schema/todo/collection.json
  Todo-Message: !include resources/schema/todo/message.json
```

- **connection**

  Provides connections to a remote service i.e. mysql or mongodb. This connection can be used inside an action:

```
connection:
  Default-Connection:
    class: Fusio\Adapter\Sql\Connection\SqlAdvanced
    config:
      url: "sqlite:///${dir.cache}/todo-app.db"
```

- **migration**

  Through migrations it is possible to execute i.e. sql queries on a connection. This allows you to change your database schema on deployment.

```
migration:
  Default-Connection:
    - resources/sql/v1_schema.php
```

Through the command `php bin/fusio deploy` you can deploy the API. It is now possible to visit the API endpoint at: `/todo`.

Access a non-public API endpoint

The POST method of the todo API is not public, because of this you need an access token in order to send a POST request.

- **Create a scope**

  At first we must create a scope for the `/todo` API endpoint. Therefor login to the backend an go to the scope panel. Click on the plus button and create a new scope `todo` which has the `/todo` route assigned.

- **Assign the scope to your user**

  In order to use a scope, the scope must be assigned to your user account. Therefor go to the user panel click on the edit button and assign the `todo` scope to your user.

- **Request a JWT**

  Now you can obtain a JWT through a simple HTTP request to the `consumer/login` endpoint.

```
POST /consumer/login HTTP/1.1
Host: 127.0.0.1
Content-Type: application/json

{
  "username": "[username]",
  "password": "[password]"
}
```

  Which returns a token i.e.:

```
{
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
↪eyJzdWIiOiI5N2JkNDUzYjdlMDZlOWFlMDQxNi00YmY2MWFiYjg4MDJjZmRmOWZmN2UyNDg4OTNmNzYyYmU5Njc5MGUzYT
↪T49Af5wnPIFYbPer3rOn-KV5PcN0FLcBVykUMCIAuwI"
}
```

- **Request the non-public API endpoint**

  Now we can use the JWT as Bearer token in the Authorization header.

```
POST /todo HTTP/1.1
Host: 127.0.0.1
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
↪eyJzdWIiOiI5N2JkNDUzYjdlMDZlOWFlMDQxNi00YmY2MWFiYjg4MDJjZmRmOWZmN2UyNDg4OTNmNzYyYmU5Njc5MGUzYT
↪T49Af5wnPIFYbPer3rOn-KV5PcN0FLcBVykUMCIAuwI
Content-Type: application/json

{
  "title": "lorem ipsum",
  "content": "lorem ipsum"
}
```

Deploy

The `.fusio.yml` deploy configuration is the main configuration file to develop an API with Fusio. This chapter explains in detail the format.

routes

A route is the rule which redirects the incoming request to an action. If a request arrives the first route which matches is used. In order to be able to evolve an API it is possible to add multiple versions for the same route. For each version it is possible to specify the allowed request methods. Each method describes the request and response schema and the action which is executed upon request. If a request method is public it is possible to request the API endpoint without an access token.

```
routes:
  "/todo":
    version: 1
    methods:
      GET:
        public: true
        response: Todo-Collection
        action: "${dir.src}/Todo/collection.php"
      POST:
        public: false
        request: Todo
        response: Todo-Message
        action: "${dir.src}/Todo/insert.php"
  "/todo/:todo_id":
    version: 1
    methods:
      GET:
        public: true
        response: Todo
```

```
      action: "${dir.src}/Todo/row.php"
    DELETE:
      public: false
      response: Todo-Message
      action: "${dir.src}/Todo/delete.php"
```

The `request` and `response` key reference a schema name which was defined under the `schema` key. It is also possible to use the `Passthru` schema which simply redirects all data. The `action` key reference an action. How this action is used depends on the `fusio_engine` setting in the `configuration.php` file. By default we use the `PhpFile` engine which uses a simple php file. But it is also possible to use a `PhpClass` or `V8` engine.

Path

The path can contain variable path fragments. It is possible to access these variable path fragments inside an action. The following list describes the syntax.

- `/news` No variable path fragment only the request to `/news` matches this route
- `/news/:news_id` Simple variable path fragment. This route matches to any value except a slash. I.e. `/news/foo` or `/news/12` matches this route
- `/news/$year<[0-9]+>` Variable path fragment with a regular expression. I.e. only `/news/2015` matches this route
- `/file/*path` Variable path fragment which matches all values. I.e. `/file/foo/bar` or `/file/12` matches this route

Status

Beside the `version` every route can also have a `status` field. By default the status is set to 4 (Development). If you change the status to 1 (Production) it is not longer possible to change the API endpoint through the backend. The following list describes each status

- `4 = Development` Used as first status to develop a new API endpoint. It adds a "Warning" header to each response that the API is in development mode.
- `1 = Production` Used if the API is ready for production use. If the API transitions from development to production all databases settings are copied into the route. That means changing a schema or action will not change the API endpoint.
- `2 = Deprecated` Used if you want to deprecate a specific version of the API. Adds a "Warning" header to each response that the API is deprecated.
- `3 = Closed` Used if you dont want to support a specific version anymore. Returns an error message with a `410 Gone` status code

Action

The action contains the business logic of your API endpoint. It i.e. selects or inserts entries from a database or pushes a new entry to a message queue.

schema

The schema defines the format of the request and response data. It uses the JsonSchema format. It is recommended to place the schemas into a separate folder and include them in the config.

```
schema:
  Todo: !include resources/schema/todo/entity.json
  Todo-Collection: !include resources/schema/todo/collection.json
  Todo-Message: !include resources/schema/todo/message.json
```

Inside a schema it is possible to refer to other schema definitions by using the `$ref` key and the `file` protocol i.e. `file:///[file]`.

```
{
    "id": "http://acme.com/schema",
    "type": "object",
    "title": "schema",
    "properties": {
        "name": {
            "type": "string"
        },
        "author": {
            "$ref": "file:///author.json"
        },
        "date": {
            "type": "string",
            "format": "date-time"
        }
    }
}
```

connection

A connection provides a class which helps to connect to another service.

```
Acme-Mysql:
  class: Fusio\Adapter\Sql\Connection\Sql
  config:
    type: pdo_mysql
    host: localhost
    username: root
    password:
    database: fusio
```

The following list contains connection classes which you can use. Note some connections depend on PHP extensions or other client libraries, you have to install the fitting adapter in order to use the connection. Take a look at the http://www.fusio-project.org/adapter website for an overview of available adapters.

Sql

Connects to a SQL database using the doctrine DBAL library.

Class `Fusio\Adapter\Sql\Connection\Sql`

Return `Doctrine\DBAL\Connection`

Website http://www.doctrine-project.org/projects/dbal.html

API http://www.doctrine-project.org/api/dbal/2.5/class-Doctrine.DBAL.Connection.html

config

type The driver which is used to connect to the database

- `pdo_mysql` = MySQL
- `pdo_pgsql` = PostgreSQL
- `sqlsrv` = Microsoft SQL Server
- `oci8` = Oracle Database
- `sqlanywhere` = SAP Sybase SQL Anywhere

host The IP or hostname of the database server

username The name of the database user

password The password of the database user

database The name of the database which is used upon connection

MongoDB

Connects to a MongoDB using the official MongoDB library. Note this requires the PHP `mongodb` extension.

Class `Fusio\Adapter\Mongodb\Connection\MongoDB`

Return `MongoDB\Database`

Website https://github.com/mongodb/mongo-php-library

API https://docs.mongodb.com/php-library/master/reference/class/MongoDBDatabase/

config

url The url must have the following format `mongodb://[username:password@]host1[:port1][, host2[:port2:],...]/db`

options It is possible to provide option parameters. The options must be url encoded i.e. `connect=1&fsync=1`

database The name of the database which is used upon connection

HTTP

Uses the Guzzle library to send HTTP requests.

Class `Fusio\Adapter\Http\Connection\Http`

Return `GuzzleHttp\Client`

Website http://docs.guzzlephp.org/en/latest/

config

url HTTP base url

username Optional username for authentication

password Optional password for authentication

proxy Optional HTTP proxy

AMQP

Provides a client to send messages to a RabbitMQ.

Class `Fusio\Adapter\Amqp\Connection\Amqp`

Return `PhpAmqpLib\Connection\AMQPStreamConnection`

Website https://github.com/php-amqplib/php-amqplib

config

`host` The IP or hostname of the RabbitMQ server

`port` The port used to connect to the AMQP broker. The port default is 5672

`user` The login string used to authenticate with the AMQP broker

`password` The password string used to authenticate with the AMQP broker

`vhost` The virtual host to use on the AMQP broker

Beanstalk

Provides a client to send messages to a Beanstalkd.

Class `Fusio\Adapter\Beanstalk\Connection\Beanstalk`

Return `Pheanstalk\Pheanstalk`

Website https://github.com/pda/pheanstalk

config

`host` The IP or hostname of the Beanstalk server

`port` Optional the port of the Beanstalk server

Cassandra

Connects to a Cassandra database using the official PHP library. Requires the `cassandra` PHP extension.

Class `Fusio\Adapter\Cassandra\Connection\Cassandra`

Return `Cassandra\Session`

Website https://github.com/datastax/php-driver

API http://datastax.github.io/php-driver/api/Cassandra/interface.Session/

config

`host` Configures the initial endpoints. Note that the driver will automatically discover and connect to the rest of the cluster

`port` Specify a different port to be used when connecting to the cluster

`keyspace` Optional keyspace name

Elasticsearch

Connects to a Elasticsearch database using the official PHP library.

Class `Fusio\Adapter\Elasticsearch\Connection\Elasticsearch`

Return `Elasticsearch\Client`

Website https://github.com/elastic/elasticsearch-php

config

`host` Comma separated list of elasticsearch hosts i.e. `192.168.1.1:9200,192.168.1.2`

Memcache

Uses the native PHP `memcached` extension to connect to a memcache server.

Class `Fusio\Adapter\Memcache\Connection\Memcache`

Return `Memcached`

Website http://php.net/manual/de/book.memcached.php

config

`host` Comma seperated list of [ip]:[port] i.e. `192.168.2.18:11211,192.168.2.19:11211`

Neo4j

Connects to a Neo7j graph database using the official PHP library.

Class `Fusio\Adapter\Neo4j\Connection\Neo4j`

Return `GraphAware\Neo4j\Client\ClientInterface`

Website https://github.com/graphaware/neo4j-php-client

config

`uri` URI of the connection i.e. `http://neo4j:password@localhost:7474`

SOAP

Provides a client to send SOAP requests.

Class `Fusio\Adapter\Soap\Connection\Soap`

Return `SoapClient`

Website http://php.net/manual/de/class.soapclient.php

config

`wsdl` Location of the WSDL specification

`location` Required if no WSDL is available

`uri` Required if no WSDL is available

`version` Optional SOAP version

- `1` = SOAP 1.1

- `2` = SOAP 1.2

`username` Optional username for authentication

`password` Optional password for authentication

migration

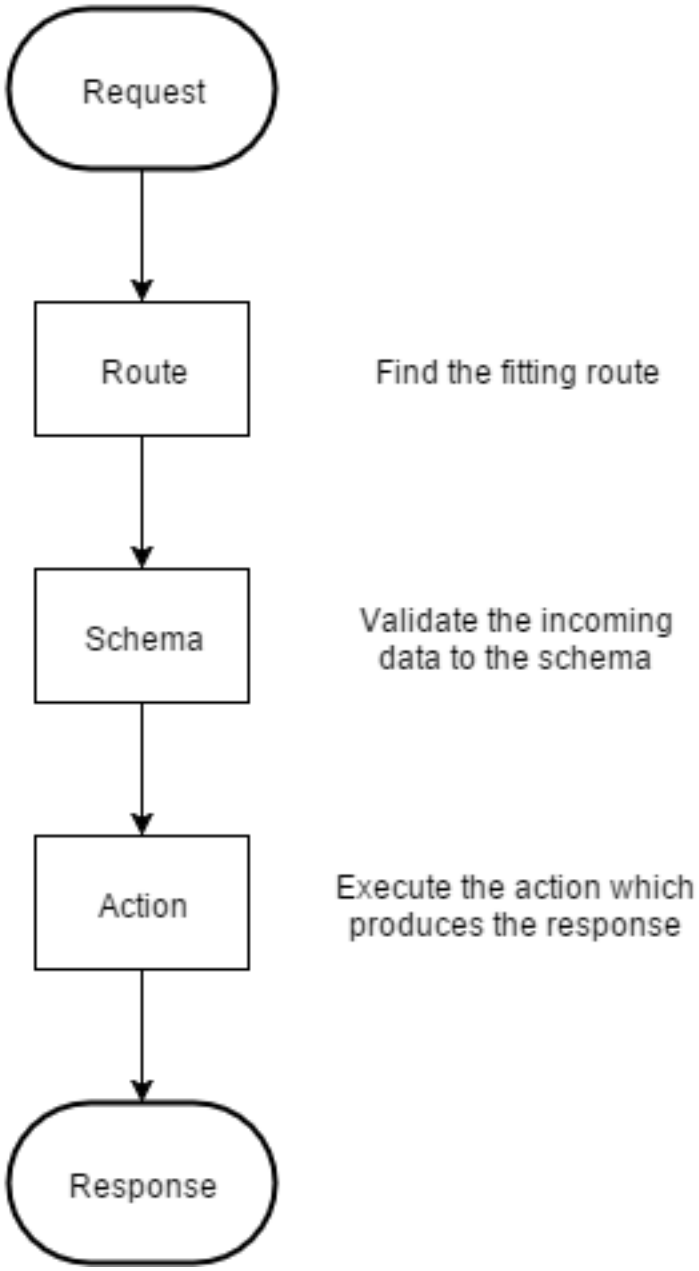
The migration key can contain an array of files per connection. The files are executed once on deployment.

```
migration:
  Default-Connection:
    - resources/sql/v1_schema.php
```

Note: If you migrate a schema to a specific database the migration tool will delete all tables from the database to adjust the tables according to the defined schema.

Request lifecycle

To get a better understanding how Fusio works lets take a look at the following flow chart:



If a request arrives Fusio looks at the routes table to find the fitting route. The route has all information which request methods are allowed, how the request and response schema is and what action to execute. The schema is a specification of the request or response data in the JSONSchema format. It is not required to specify a schema for your endpoint but it is recommended since the documentation is based on the schema. If the request arrives at the action the business logic of the endpoint is executed. A action can use connections in order to connect to a remote service and execute a specific task. In case the route is protected the consumer has to authorize the request through OAuth2. Therefor he needs a consumer key and secret which he can obtain through an app. Each app has specific scopes assigned so that the app can only access the specific parts of the API.

Backend

This chapter contains information about the backend. The following pages are extracted from the online help which is also available inside the backend app:

Action

```
## Action

The action contains the logic to handle the request and produce a response. Each
action is based on a class and can have specific parameters. Fusio contains
already some actions for common tasks i.e. to execute database operations or
push data to a message queue. It is also possible to provide a custom
implementation. Please take a look at the `adapter` chapter for more
information how to build a custom implementation.
```

App

```
## App

An app enables the consumer to request an access token through the app key and
secret. With the access token it is possible to request protected API
endpoints. There is a default consumer implementation located at `developer/`
which enables a user to manage their apps. The consumer can use any OAuth2 client
to request an access token. Fusio supports by default the `authorization_code`,
`implicit` and `password` grant type. More detailed informations about the
[OAuth2] flow.

### Authorization code

At first you have to redirect the client to the consumer endpoint containing
the app key, redirect uri and the needed scopes i.e.:
`/developer/auth?response_type=code&client_id=[key]&redirect_uri=[url]&scope=foo,bar`.
After the user has authenticated he approves or denies the access. If he accepts
the user gets redirected to the provided `redirect_uri`. Note the `redirect_uri`
must have the same host as the url which was provided for the app. The callback
contains a GET parameter `code` which can be [exchanged] for an access token at
the `/authorization/token` endpoint.

### Implicit

Mostly used for javascript apps. Like in the authorization code flow the app
redirects the user to the consumer endpoint i.e.:
`/developer/auth?response_type=token&client_id=[key]&redirect_uri=[url]&scope=foo,bar`
If the user has authenticated and approved the app the user gets redirected to
the `redirect_uri`. The callback contains the access token in the [fragment]
component. The access tokens which are issued through the implicit grant have
usually a much shorter life time because they are more insecure. It is also
possible to deactivate the implicit grant through the configuration.

### Password
```

```
A user can use the password grant to obtain directly an access token with
their username and password. Therefor he has to send a [direct] request to the
`/authorization/token` endpoint.


[OAuth2]: https://tools.ietf.org/html/rfc6749
[exchanged]: https://tools.ietf.org/html/rfc6749#section-4.1.3
[direct]: https://tools.ietf.org/html/rfc6749#section-4.3.2
[fragment]: https://tools.ietf.org/html/rfc6749#section-4.2.2
```

Config

```
## Config

The config contains system wide settings. In the following we explain some
important settings which you most likely need to configure.

* `mail_register_body`
  If a new user registers through the consumer app he receives an activation
  mail. Through this setting you can configure the text and adjust the
  activtaion url
* `recaptcha_secret`
  If provided the consumer registration can show an google recaptcha which
  prevents automatic registration. You also have to provide the recaptcha public
  key to the consumer app
* `scopes_default`
  Those are the scopes which are assigned by default if a new user registers
* `provider_facebook_secret` `provider_github_secret` `provider_google_secret`
  If provided a user can login through those remote providers. You also have
  to provide the app key to the consumer app
```

Connection

```
## Connection

A connection enables Fusio to connect to other remote sources. This can be i.e.
a database or message queue server. Please take a look at the `adapter` chapter
for more information how to build a custom implementation.
```

Import

```
## Import

The importer provides a way to import route and schema definitions. The data
must be in the [RAML] or [Swagger] format. The importer displays a preview what
data is imported before any changes are made.
```

```
[RAML]: http://raml.org/
[Swagger]: http://swagger.io/
```

Rate

```
## Rate

Through a rate it is possible to limit the amount of incoming requests to a
threshold. If the threshold is reached the user receives a 429 http status code.
A rate can distinguish between authenticated and not authenticated calls. For
authenticated calls the request count is based on the app for not authenticated
calls it is based on the ip address.
```

Routes

```
## Routes

A route is the rule which redirects the incoming request to an action. If a
request arrives the first route which matches is used. In order to be able to
evolve an API it is possible to add multiple versions for the same route. For
each version it is possible to specify the allowed request methods. Each method
describes the request and response schema and the action which is executed upon
request. If a request method is public it is possible to request the API
endpoint without an access token.

### Path

The path can contain variable path fragments. It is possible to access these
variable path fragments inside an action. The following list describes the
syntax.

* `/news`
  No variable path fragment only the request to `/news` matches this route

* `/news/:news_id`
  Simple variable path fragment. This route matches to any value except a slash.
  I.e. `/news/foo` or `/news/12` matches this route

* `/news/$year<[0-9]+>`
  Variable path fragment with a regular expression. I.e. only `/news/2015`
  matches this route

* `/file/*path`
  Variable path fragment which matches all values. I.e. `/file/foo/bar` or
  `/file/12` matches this route

### Status

The status affects the behaviour of the API endpoint. The following list
describes each status
```

```
* `Development`
  Used as first status to develop a new API endpoint. It adds a "Warning" header
  to each response that the API is in development mode.

* `Production`
  Used if the API is ready for production use. If the API transitions from
  development to production all databases settings are copied into the route.
  That means changing a schema or action will not change the API endpoint.

* `Deprecated`
  Used if you want to deprecate a specific version of the API. Adds a "Warning"
  header to each response that the API is deprecated.

* `Closed`
  Used if you dont want to support a specific version anymore. Returns an error
  message with a `410 Gone` status code

### Action

The action contains the business logic of your API endpoint. It i.e. selects
or inserts entries from a database or pushes a new entry to a message queue.
```

Schema

```
## Schema

The schema defines the format of the request and response data. It uses the
[JsonSchema] format. Inside a schema it is possible to refer to other schema
definitions by using the <code>$ref</code> key and the <code>schema</code>
protocol i.e. <code>schema:///[schema-name]</code>. More detailed information
about the json schema format at the [RFC].

### Example

    {
        "id": "http://acme.com/schema",
        "type": "object",
        "title": "schema",
        "properties": {
            "name": {
                "type": "string"
            },
            "author": {
                "$ref": "schema:///author"
            },
            "date": {
                "type": "string",
                "format": "date-time"
            }
        }
    }


[JsonSchema]: http://json-schema.org/
```

```
[RFC]: http://tools.ietf.org/html/draft-zyp-json-schema-04
```

Scope

```
## Scope

A scope describes the right to access specific routes and request methods. Each
user account has assigned a set of allowed scopes. If a user creates an app he
can only assign the scopes which are available for him.
```

User

```
## User

A user is either a `Consumer` which uses the API or an `Administrator` which
manages the API through the backend. An Administrator account can request an
access token for the backend API. Fusio has a simple consumer backend located
at `/developer` where a user can manage all app settings.
```

Consumer

Fusio provides a default consumer implementation located at `/developer` which provides a basic admin panel to manage and authorize apps. It is also possible to integrate it into an existing application. In the following an explanation how to authorize an app.

Authorization code

At first you have to redirect the client to the consumer endpoint containing the app key, redirect uri and the needed scopes i.e.: `/developer/auth?response_type=code&client_id=[key]&redirect_uri=[url]&scope=foo,bar`. After the user has authenticated he approves or denies the access. If he accepts the user gets redirected to the provided `redirect_uri`. Note the `redirect_uri` must have the same host as the url which was provided for the app. The callback contains a GET parameter `code` which can be exchanged for an access token at the `/authorization/token` endpoint.

Implicit

Mostly used for javascript apps. Like in the authorization code flow the app redirects the user to the consumer endpoint i.e.: `/developer/auth?response_type=token&client_id=[key]&redirect_uri=[url]&scope=foo,bar` If the user has authenticated and approved the app the user gets redirected to the `redirect_uri`. The callback contains the access token in the fragment component. The access tokens which are issued through the implicit grant have usually a much shorter life time because they are more insecure. It is also possible to deactivate the implicit grant through the configuration.

Password

A user can use the password grant to obtain directly an access token with their username and password. Therefor he has to send a direct request to the `/authorization/token` endpoint.

Adapter

An adapter is a composer package which provides classes to extend the functionality of Fusio. Through an adapter it is i.e. possible to provide custom action/connection classes or to install predefined routes for an existing system. A package needs to require the `fusio/engine` package and must have an adapter class which implements the `Fusio\Engine\AdapterInterface` class. This class has a method `getDefinition` which returns an absolute path to a `adapter.json` definition. This definition contains all information for Fusio how to extend the system. The adapter can be installed through the register command:

```
php bin/fusio system:register Acme\System\Adapter
```

In the following an example adapter definition which showcases all available parameters.

```
{
    "actionClass": ["Fusio\\Impl\\Tests\\Adapter\\Test\\VoidAction"],
    "connectionClass": ["Fusio\\Impl\\Tests\\Adapter\\Test\\VoidConnection"],
    "routes": [{
        "path": "/void",
        "config": [{
            "version": 1,
            "status": 4,
            "methods": {
                "GET": {
                    "active": true,
                    "public": true,
                    "action": "Void-Action",
                    "request": "Adapter-Schema",
                    "response": "Passthru"
                }
            }
        }]
    }],
    "action": [{
        "name": "Void-Action",
        "class": "Fusio\\Impl\\Tests\\Adapter\\Test\\VoidAction",
        "config": {
            "foo": "bar",
            "connection": "Adapter-Connection"
        }
    }],
    "schema": [{
        "name": "Adapter-Schema",
        "source": {
            "id": "http://fusio-project.org",
            "title": "process",
            "type": "object",
            "properties": {
                "logId": {
                    "type": "integer"
                },
                "title": {
```

```
                    "type": "string"
                },
                "content": {
                    "type": "string"
                }
            }
        }
    }],
    "connection": [{
        "name": "Adapter-Connection",
        "class": "Fusio\\Impl\\Tests\\Adapter\\Test\\VoidConnection",
        "config": {
            "foo": "bar"
        }
    }]
}
```

It is also possible to generate such a definition on an existing system through the `system:export` command.

```
php bin/fusio system:export > export.json
```

CHAPTER 2

Help

Because Fusio is in an early stage the manual is not complete. We appreciate every help in making this documentation better.