Fusio Documentation

Release 1.0

Christoph Kappestein

Sep 23, 2018

Contents

Fusio is an open source API management platform which helps to build and manage RESTful APIs. The documentation covers the basic steps how to develop and maintain an API with Fusio.

CHAPTER 1

Overview

1.1 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, develop SPAs or Mobile-Apps. Because of this we think that Fusio is a great tool to simplify building such APIs. More information on https://www.fusio-project.org/

1.2 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.

- **Logging**

  All errors which occur in your endpoint are logged and are visible at the backend including all information from the request.

- **Connection**

  Fusio provides an adapter system to connect to external services. By default we provide the HTTP and SQL connection type but there are many other types available i.e. MongoDB, Amqp, Cassandra.

- **Migration**

  Fusio has a migration system which allows you to change the database schema on deployment.

- **Testing**

  Fusio provides an api test case wherewith you can test every endpoint response without setting up a local web server.

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

1.3 Development

Fusio provides two ways to develop an API. The first way is to build API endpoints only through the backend interface by using all available actions. Through this you can solve already many tasks especially through the usage of the v8 action.

The other way is to use the deploy mechanism. Through this you can use normal PHP files to implement your business logic and thus you have ability to use the complete PHP ecosystem. Therefor you need to define a `.fusio.yml` deploy file which specifies the available routes and actions of the system. This file can be deployed with the following command:

```
php bin/fusio deploy
```

The action of each route contains the source which handles the business logic. This can be i.e. a simple php file, php class or a url. 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
```

(continues on next page)

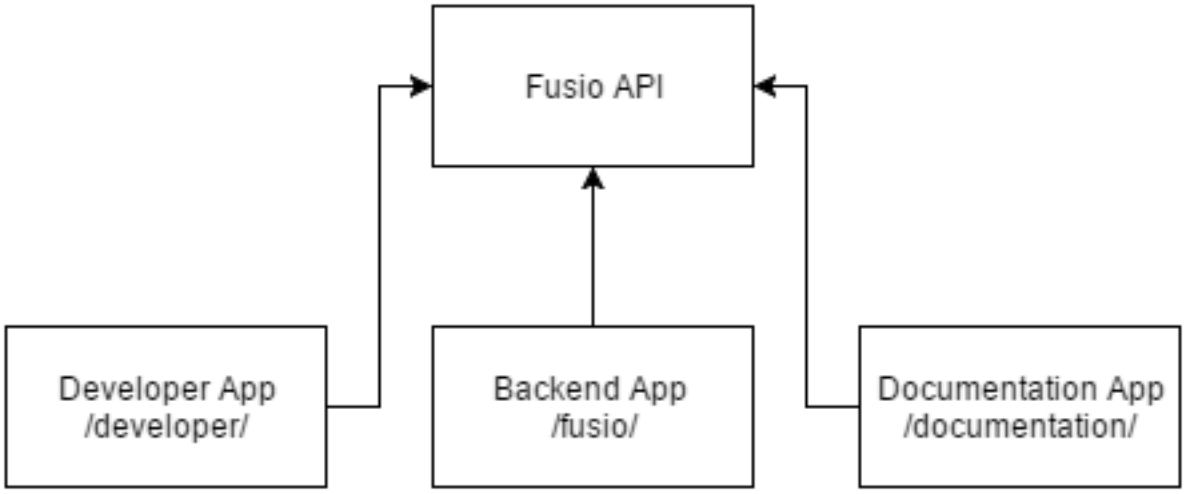
(continued from previous page)

```
 */

/** @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.

1.4 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:



1.4.1 API

If you install a Fusio system it setups the default API. Through the API 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 the system configuration

- `/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

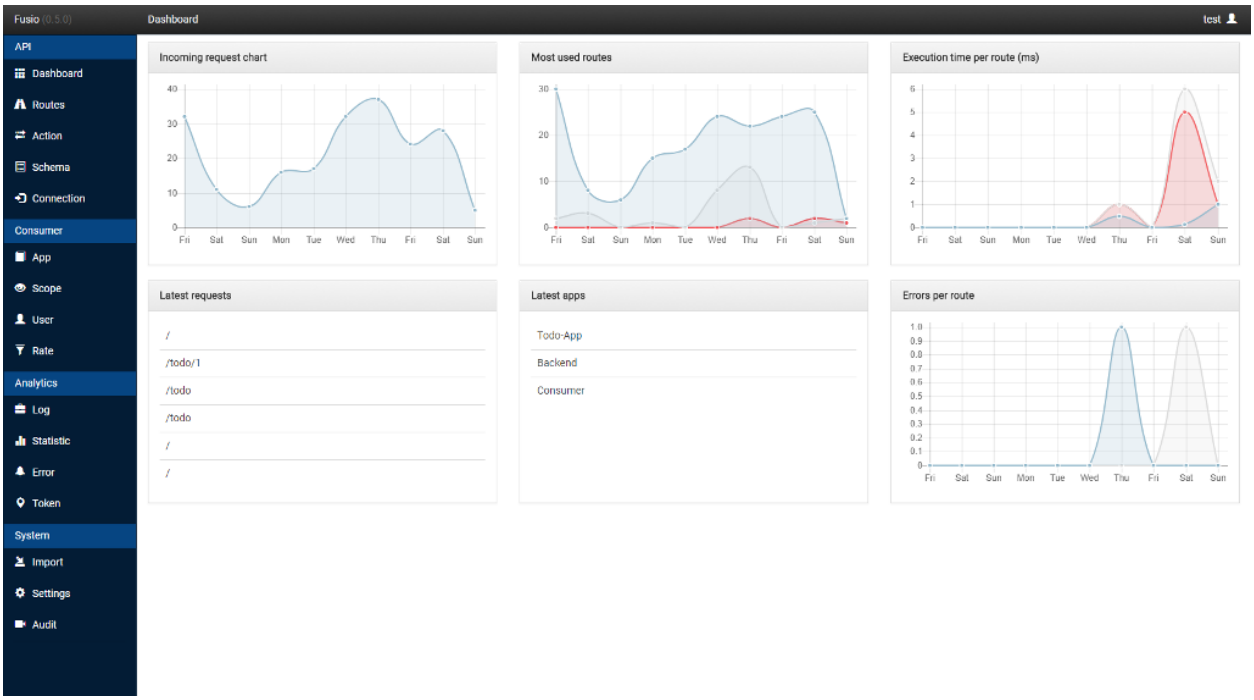
- `/export`

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

1.5 Apps

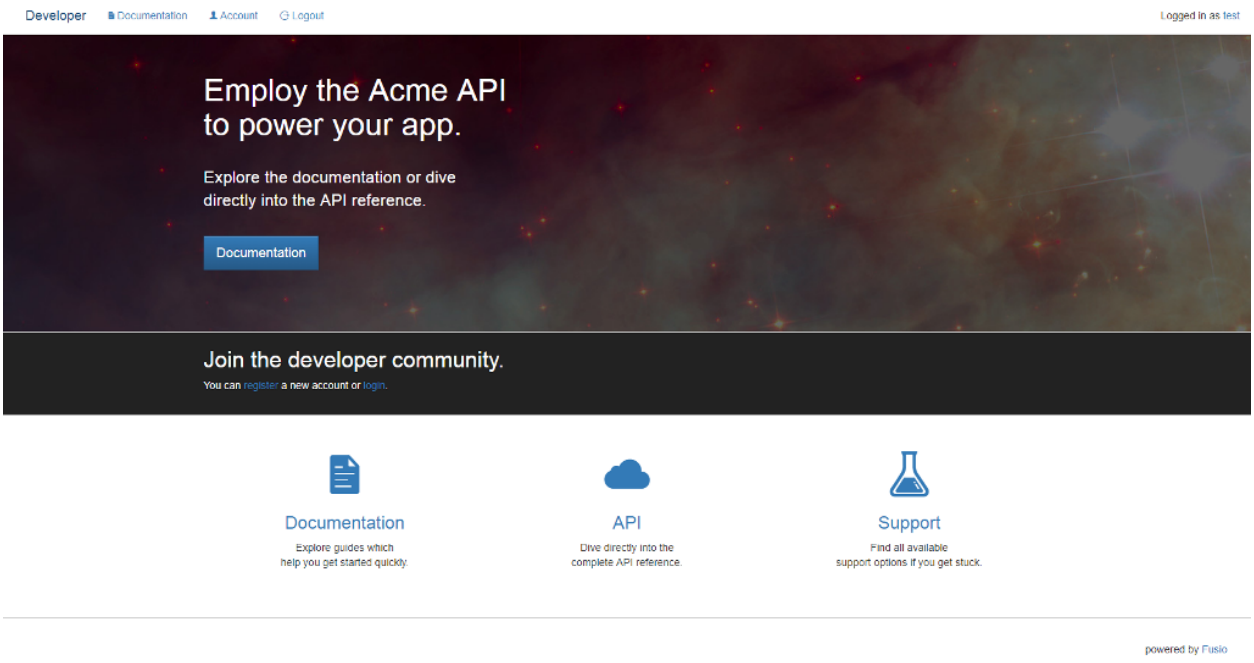
The following apps are working with the Fusio API.

1.5.1 Backend



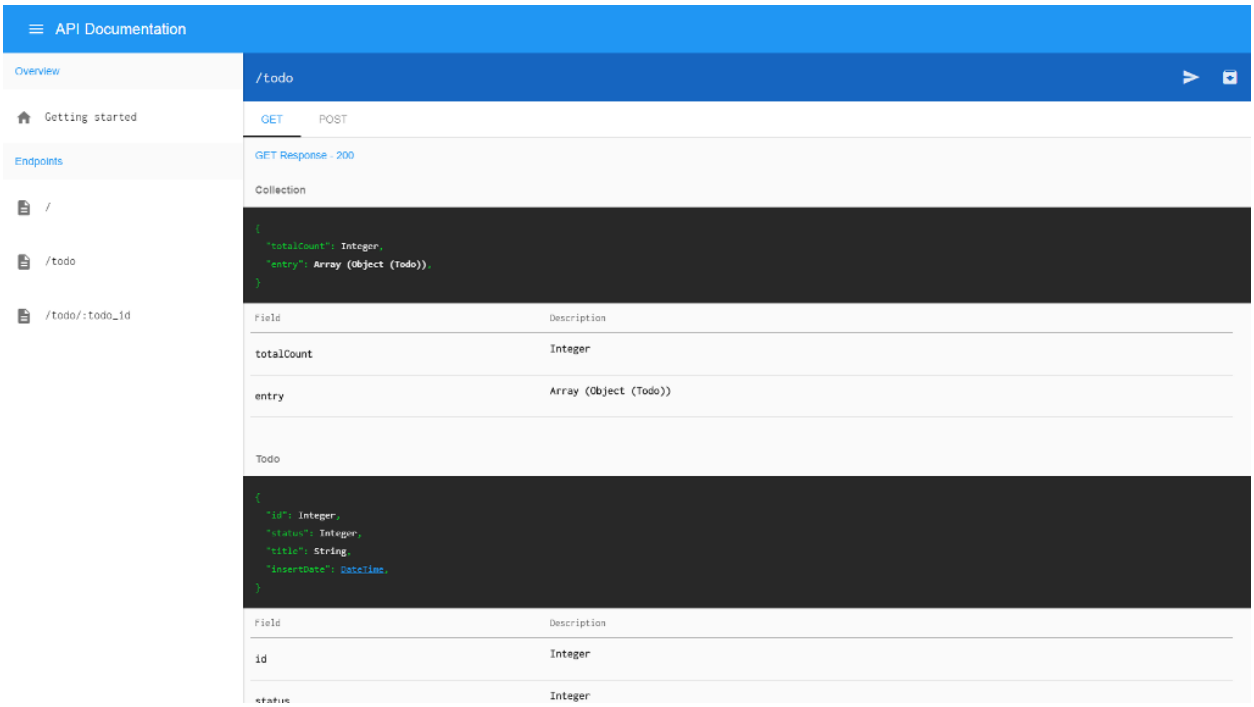
The backend app is the app where the administrator can configure the system. The app is located at `/fusio/`.

1.5.2 Developer



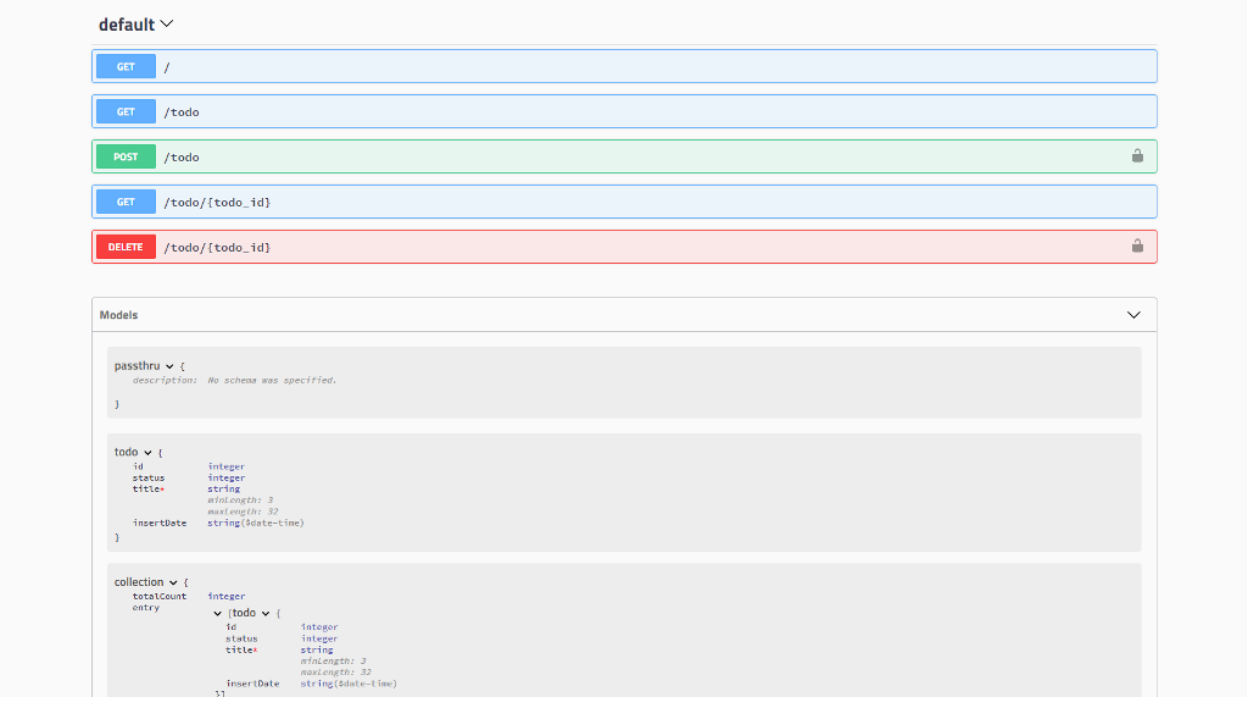
The developer app is designed to quickly setup an API program where new developers can register and create/manage their apps. The app is located at `/developer/`.

1.5.3 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/`.

1.5.4 Swagger-UI



The swagger-ui app renders a documentation based on the OpenAPI specification. The app is located at */swagger-ui/*.

CHAPTER 2

Installation

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

Composer

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

Download

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

2.1 Configuration

You can either manually install Fusio with the steps below or you can also use the browser based installer at `public/install.php`. Note because of security reasons it is highly recommended to remove the installer script after the installation.

- **Adjust the configuration file**

  Open the file `.env` in the Fusio directory and change the key `FUSIO_URL` to the domain pointing to the public folder. Also insert the database credentials to the `FUSIO_DB_*` 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/`.

2.2 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.

2.3 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',
```

2.3.1 Apache

```
<VirtualHost *:80>
    ServerName api.acme.com
    DocumentRoot /var/www/html/fusio/public

    <Directory /var/www/html/fusio/public>
        Options FollowSymLinks
        AllowOverride All
        Require all granted

        # rewrite
        RewriteEngine On
        RewriteBase /

        RewriteCond %{REQUEST_FILENAME} !-f
        RewriteCond %{REQUEST_FILENAME} !-d
        RewriteRule (.*) /index.php/$1 [L]

        RewriteCond %{HTTP:Authorization} ^(.*)
        RewriteRule .* - [e=HTTP_AUTHORIZATION:%1]
    </Directory>

    # log
    LogLevel warn
    ErrorLog ${APACHE_LOG_DIR}/fusio.error.log
    CustomLog ${APACHE_LOG_DIR}/fusio.access.log combined
</VirtualHost>
```

You should enable the module `mod_rewrite` so that the .htaccess file in the public folder is used. It is also possible to include the htaccess commands directly into the virtual host which also increases performance. 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`.

2.3.2 Shared-Hosting

If you want to run Fusio on a shared-hosting environment it is possible but in general not recommended since you can not properly configure the web server and access the CLI. Therefore you can not use the deploy command which simplifies development. The biggest problem of a shared hosting environment is that you can not set the document root to the `public/` folder. If you place the following `.htaccess` file in the directory you can bypass this problem by redirecting all requests to the `public/` folder.

```
RewriteEngine on
RewriteRule (.*) public/$1/
```

While this may work many shared hosting provider have strict limitations of specific PHP functions which are maybe used by Fusio and which produce other errors.

2.3.3 cPanel

On cPanel you can create a new sub-domain and use the "base document path" option to point it at `/public`.

2.4 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

2.5 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.

2.5.1 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.

2.6 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.

2.6.1 Server

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.

2.6.2 App

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

- `public/fusio/`

CHAPTER 3

Get started

3.1 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:

```
"/todo": !include resources/routes/todo/collection.yaml
"/todo/:todo_id": !include resources/routes/todo/entity.yaml
```

- **schema**

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

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

- **connection**

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

```
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.

```
Default-Connection:
  - resources/migration/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`.

3.2 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.

- **Assign the scope to your user**

  By default all routes are assigned to the `todo` scope. 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. It is also possible to set the default scopes for new users under settings `scopes_default`.

- **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"
}
```

  Note this generates an OAuth2 token with contains all scopes from your user account. It is also possible to use the OAuth2 endpoint */authorization/token* to create an access token with specific assigned scopes.

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

  Now we can use the JWT as Bearer token in the `Authorization` header to access the protected endpoint.

```
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"
}
```

CHAPTER 4

Deploy

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

4.1 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.

```
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"
```

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.

4.1.1 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

4.1.2 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

4.2 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 `$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"
        }
    }
}
```

4.3 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: test
    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.

4.3.1 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

4.3.2 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

4.3.3 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

4.3.4 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

4.3.5 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

4.3.6 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

4.3.7 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`

4.3.8 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`

4.3.9 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`

4.3.10 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

4.4 migration

The migration key can contain an array of files per connection. The files are executed once on deployment. At the moment migrations are only supported for SQL connections.

```
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 This means all tables which are not defined in the migration file will be deleted.

CHAPTER 5

Development

This chapter contains information how to develop endpoint actions.

5.1 Action

The `src/` folder contains the action code which is executed if a request arrives at an endpoint which was specified in the `.fusio.yml` deploy file. Fusio determines the engine based on the provided action string. The following engines are available:

5.1.1 Engines

PHP File

```
action: "${dir.src}/Todo/collection.php"
```

If the action points to a file with a `php` file extension Fusio simply includes this file. In the following an example implementation:

```
<?php
/**
 * @var \Fusio\Engine\ConnectorInterface $connector
 * @var \Fusio\Engine\ContextInterface $context
 * @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
 */

// @TODO handle request and return response

```

(continues on next page)

(continued from previous page)

```
$response->build(200, [], [
    'message' => 'Hello World!',
]);
```

Javascript File

```
action: "${dir.src}/Todo/collection.js"
```

If the action points to a file with a `js` file extension Fusio uses the internal v8 engine to execute the js code. This is suitable for javascript developers who like to write the code in javascript. Note the v8 implementation requires the php-v8 extension. In the following an example implementation:

```
response.setStatusCode(200);
response.setBody({
    message: "Hello World!"
});
```

HTTP Url

```
action: "http://foo.bar"
```

If the action contains an `http` or `https` url the request gets forwarded to the defined endpoint. Fusio automatically adds some additional headers to the request which may be used by the endpoint i.e.:

```
X-Fusio-Route-Id: 72
X-Fusio-User-Anonymous: 1
X-Fusio-User-Id: 4
X-Fusio-App-Id: 3
X-Fusio-App-Key: 1ba7b2e5-fa1a-4153-8668-8a855902edda
X-Fusio-Remote-Ip: 127.0.0.1
```

Static file

```
action: "${dir.src}/static.json"
```

If the action points to a simple file Fusio will simply forward the content to the client. This is helpful if you want to build fast an sample API with dummy responses.

PHP Class

```
action: "App\\Todo\\CollectionAction"
```

If the action string is a PHP class Fusio tries to autoload this class through composer. The class must implement the `Fusio\Engine\ActionInterface`. This is the most advanced solution since it is also possible to access services from the DI container. In the following an example implementation:

```php
<?php

namespace App\Todo;

use Fusio\Engine\ActionAbstract;
use Fusio\Engine\ContextInterface;
use Fusio\Engine\ParametersInterface;
use Fusio\Engine\RequestInterface;

class CollectionAction extends ActionAbstract
{
    public function handle(RequestInterface $request, ParametersInterface
↪$configuration, ContextInterface $context)
    {
        // @TODO handle request and return response

        return $this->response->build(200, [], [
            'message' => 'Hello World!',
        ]);
    }
}
```

5.1.2 Examples

We have several example actions which show how to implement a specific task as action.

SQL

Select

Routes

resources/routes.yaml

```
"/test":
  version: 1
  methods:
    GET:
      public: true
      action: "${dir.src}/sql-select.php"
```

Connection

resources/connections.yaml

```
Database-Connection:
  class: Fusio\Adapter\Sql\Connection\Sql
  config:
    type: "pdo_mysql"
    host: "127.0.0.1"
    username: "app"
```

(continues on next page)

(continued from previous page)

```
    password: "secret"
    database: "app"
```

Action

src/sql-select.php

```
<?php
/**
 * @var \Fusio\Engine\ConnectorInterface $connector
 * @var \Fusio\Engine\ContextInterface $context
 * @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('Database-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,
]);
```

Select-Row

Routes

resources/routes.yaml

```
"/test/:id":
  version: 1
  methods:
    GET:
      public: true
      action: "${dir.src}/sql-select-row.php"
```

Connection

resources/connections.yaml

```
Database-Connection:
  class: Fusio\Adapter\Sql\Connection\Sql
  config:
```

(continues on next page)

(continued from previous page)

```
    type: "pdo_mysql"
    host: "127.0.0.1"
    username: "app"
    password: "secret"
    database: "app"
```

Action

src/sql-select-row.php

```
<?php
/**
 * @var \Fusio\Engine\ConnectorInterface $connector
 * @var \Fusio\Engine\ContextInterface $context
 * @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
 */

use PSX\Http\Exception as StatusCode;

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

$todo = $connection->fetchAssoc('SELECT * FROM app_todo WHERE id = :id', [
    'id' => $request->getUriFragment('id')
]);

if (empty($todo)) {
    throw new StatusCode\NotFoundException('Entry not available');
}

return $response->build(200, [], $todo);
```

Insert

Routes

resources/routes.yaml

```
"/test":
  version: 1
  methods:
    POST:
      public: true
      action: "${dir.src}/sql-insert.php"
```

Connection

resources/connections.yaml

```
Database-Connection:
  class: Fusio\Adapter\Sql\Connection\Sql
  config:
    type: "pdo_mysql"
    host: "127.0.0.1"
    username: "app"
    password: "secret"
    database: "app"
```

Action

src/sql-insert.php

```
<?php
/**
 * @var \Fusio\Engine\ConnectorInterface $connector
 * @var \Fusio\Engine\ContextInterface $context
 * @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
 */

use PSX\Http\Exception as StatusCode;

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

$body = $request->getBody();
$now  = new \DateTime();

if (empty($body->title)) {
    throw new StatusCode\BadRequestException('No title provided');
}

$connection->insert('app_todo', [
    'status' => 1,
    'title' => $body->title,
    'insertDate' => $now->format('Y-m-d H:i:s'),
]);

return $response->build(201, [], [
    'success' => true,
    'message' => 'Insert successful',
]);
```

Update

Routes

resources/routes.yaml

```
"/test/:id":
  version: 1
  methods:
    PUT:
      public: true
      action: "${dir.src}/sql-update.php"
```

Connection

resources/connections.yaml

```
Database-Connection:
  class: Fusio\Adapter\Sql\Connection\Sql
  config:
    type: "pdo_mysql"
    host: "127.0.0.1"
    username: "app"
    password: "secret"
    database: "app"
```

Action

src/sql-update.php

```
<?php
/**
 * @var \Fusio\Engine\ConnectorInterface $connector
 * @var \Fusio\Engine\ContextInterface $context
 * @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
 */

use PSX\Http\Exception as StatusCode;

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

$body = $request->getBody();
$now  = new \DateTime();

if (empty($body->title)) {
    throw new StatusCode\BadRequestException('No title provided');
}

$affected = $connection->update('app_todo', [
    'title' => $body->title,
    'lastUpdated' => $now->format('Y-m-d H:i:s'),
], [
    'id' => $request->getUriFragment('id')
]);
```

(continues on next page)

(continued from previous page)

```
if (empty($affected)) {
    throw new StatusCode\NotFoundException('Entry not available');
}

return $response->build(200, [], [
    'success' => true,
    'message' => 'Update successful',
]);
```

Delete

Routes

resources/routes.yaml

```
"/test/:id":
  version: 1
  methods:
    DELETE:
      public: true
      action: "${dir.src}/sql-delete.php"
```

Connection

resources/connections.yaml

```
Database-Connection:
  class: Fusio\Adapter\Sql\Connection\Sql
  config:
    type: "pdo_mysql"
    host: "127.0.0.1"
    username: "app"
    password: "secret"
    database: "app"
```

Action

src/sql-delete.php

```
<?php
/**
 * @var \Fusio\Engine\ConnectorInterface $connector
 * @var \Fusio\Engine\ContextInterface $context
 * @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
 */
```

(continues on next page)

(continued from previous page)

```
use PSX\Http\Exception as StatusCode;

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

$affected = $connection->delete('app_todo', [
    'id' => $request->getUriFragment('id')
]);

if (empty($affected)) {
    throw new StatusCode\NotFoundException('Entry not available');
}

return $response->build(200, [], [
    'success' => true,
    'message' => 'Delete successful',
]);
```

MongoDB

The MongoDB connection is not available in the standard installation since it requires the ext-mongodb PHP extension. Therefor you need to install the extension and regsiter the Fusio MongoDB adapter.

```
composer require fusio/adapter-mongodb
php bin/fusio system:register "Fusio\Adapter\Mongodb\Adapter"
```

Find

Routes

`resources/routes.yaml`

```
"/test":
  version: 1
  methods:
    GET:
      public: true
      action: "${dir.src}/mongodb-find.php"
```

Connection

`resources/connections.yaml`

```
Mongodb-Connection:
  class: Fusio\Adapter\Mongodb\Connection\MongoDB
  config:
    url: "mongodb://127.0.0.1"
    database: "app"
```

Action

src/mongodb-find.php

```php
<?php
/**
 * @var \Fusio\Engine\ConnectorInterface $connector
 * @var \Fusio\Engine\ContextInterface $context
 * @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 \MongoDB\Database $connection */
$connection = $connector->getConnection('Mongodb-Connection');
$collection = $connection->selectCollection('app_todo');

$entries = $collection->find([], [
    'limit' => 5,
]);

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

Find-One

Routes

resources/routes.yaml

```
"/test/:id":
  version: 1
  methods:
    GET:
      public: true
      action: "${dir.src}/mongodb-find-one.php"
```

Connection

resources/connections.yaml

```
Mongodb-Connection:
  class: Fusio\Adapter\Mongodb\Connection\MongoDB
  config:
    url: "mongodb://127.0.0.1"
    database: "app"
```

Action

src/mongodb-find-one.php

```php
<?php
/**
 * @var \Fusio\Engine\ConnectorInterface $connector
 * @var \Fusio\Engine\ContextInterface $context
 * @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 \MongoDB\Database $connection */
$connection = $connector->getConnection('Mongodb-Connection');
$collection = $connection->selectCollection('app_todo');

$entries = $collection->findOne([
    'id' => ['$eq' => $request->getUriFragment('id')],
]);

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

Insert

Routes

resources/routes.yaml

```yaml
"/test":
  version: 1
  methods:
    POST:
      public: true
      action: "${dir.src}/mongodb-insert.php"
```

Connection

resources/connections.yaml

```yaml
Mongodb-Connection:
  class: Fusio\Adapter\Mongodb\Connection\MongoDB
  config:
    url: "mongodb://127.0.0.1"
    database: "app"
```

Action

src/mongodb-insert.php

```
<?php
/**
 * @var \Fusio\Engine\ConnectorInterface $connector
 * @var \Fusio\Engine\ContextInterface $context
 * @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 \MongoDB\Database $connection */
$connection = $connector->getConnection('Mongodb-Connection');
$collection = $connection->selectCollection('app_todo');

$collection->insertOne($request->getBody());

return $response->build(201, [], [
    'success' => true,
    'message' => 'Insert successful',
]);
```

Update

Routes

resources/routes.yaml

```
"/test/:id":
  version: 1
  methods:
    PUT:
      public: true
      action: "${dir.src}/mongodb-update.php"
```

Connection

resources/connections.yaml

```
Mongodb-Connection:
  class: Fusio\Adapter\Mongodb\Connection\MongoDB
  config:
    url: "mongodb://127.0.0.1"
    database: "app"
```

Action

src/mongodb-update.php

```php
<?php
/**
 * @var \Fusio\Engine\ConnectorInterface $connector
 * @var \Fusio\Engine\ContextInterface $context
 * @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 \MongoDB\Database $connection */
$connection = $connector->getConnection('Mongodb-Connection');
$collection = $connection->selectCollection('app_todo');

$filter = [
    'id' => ['$eq' => $request->getUriFragment('id')]
];

$collection->updateOne($filter, $request->getBody());

return $response->build(200, [], [
    'success' => true,
    'message' => 'Update successful',
]);
```

Delete

Routes

resources/routes.yaml

```yaml
"/test/:id":
  version: 1
  methods:
    DELETE:
      public: true
      action: "${dir.src}/mongodb-delete.php"
```

Connection

resources/connections.yaml

```yaml
Mongodb-Connection:
  class: Fusio\Adapter\Mongodb\Connection\MongoDB
  config:
    url: "mongodb://127.0.0.1"
    database: "app"
```

Action

src/mongodb-delete.php

```php
<?php
/**
 * @var \Fusio\Engine\ConnectorInterface $connector
 * @var \Fusio\Engine\ContextInterface $context
 * @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 \MongoDB\Database $connection */
$connection = $connector->getConnection('Mongodb-Connection');
$collection = $connection->selectCollection('app_todo');

$filter = [
    'id' => ['$eq' => $request->getUriFragment('id')]
];

$collection->updateOne($filter, $request->getBody());

return $response->build(200, [], [
    'success' => true,
    'message' => 'Update successful',
]);
```

5.2 Business logic

In case your API is not a simple CRUD app you probably need to execute some more complex business logic. This chapter shows options how you can organize this business logic for simple reuse.

In general an action should not contain any business logic it should simply forward the data to the fitting library or service. It is equivalent to a controller in a classical framework environment. If an action contains too many lines of code or you copy specific code snippets into different actions it is a smell to extract this logic into an external service. Inside an action you can then reuse this external service.

Fusio is designed to help you write framework independent code. That means that all services which you develop are complete free of Fusio specific code so you can simply reuse those components in another context.

5.2.1 Library

The simplest solution is to move business logic into a separate PHP class. This class can be autoloaded through composer. You can place this class either directly into the `src/` folder or develop a custom PHP package and require this package through composer.

In general a library should work with a specific connection. The following example shows a simple custom logger implementation which you could use in different actions.

```php
<?php

$connection = $connector->get('Mysql-1');

$myLogger = new MyLogger($connection);
$myLogger->log('A new log entry');
```

A simple implementation of the logger could look like:

```
<?php

namespace Acme\MyLib;

use Doctrine\DBAL\Connection;

class MyLogger
{
    /**
     * @var \Doctrine\DBAL\Connection
     */
    protected $connection;

    public function __construct(Connection $connection)
    {
        $this->connection = $connection;
    }

    public function log($message)
    {
        $this->connection->insert('my_table', [
            'message' => $message,
        ]);
    }
}
```

5.2.2 Microservice

If your business logic is more complex or has specific performance requirements you could also develop it as an external microservice. This has the advantage that service is completely decoupled from your app and it is also possible to use a complete different language. Usually you can talk to the micro service through HTTP. But it would be also possible to use a different protocol i.e. an AMQP connection to use a message queue.

```
<?php

$client = $connector->get('Http-1');

$myClient = new MyClient($client);
$myClient->send(['foo' => 'bar']);
```

A simple client implementation could look like:

```
<?php

namespace Acme\MyLib;

use GuzzleHttp\Client;

class MyClient
{
    /**
     * @var \GuzzleHttp\Client
     */
    protected $client;
```

(continues on next page)

(continued from previous page)

```
    public function __construct(Client $client)
    {
        $this->client = $client;
    }

    public function send($data)
    {
        $this->client->post('http://foo.bar/my_service', [
            'json' => $data
        ]);
    }
}
```

5.2.3 DI Container

Fusio uses a DI container to manage all internal services. You can also use this internal DI container in your action to access Fusio specific functions. It is also possible to extend the container with custom services. There for you need to add your service to the `container.php` file:

```
<?php

$container = new Fusio\Impl\Dependency\Container();
$container->setParameter('config.file', __DIR__ . '/configuration.php');

$container->set('my_service', function($c){
    return new MyService();
});

return $container;
```

To access this service in your action you need to use the following PHP action class. Note we do not recommend to rely heavily on the DI container instead use the technique describe in the chapter above to develop platform independent services which can be reused across multiple actions and applications.

```
<?php

namespace App;

use Fusio\Engine\ActionAbstract;
use Fusio\Engine\ContextInterface;
use Fusio\Engine\ParametersInterface;
use Fusio\Engine\RequestInterface;
use Fusio\Engine\Factory\ContainerAwareInterface;
use Psr\Container\ContainerInterface;

class Endpoint extends ActionAbstract implements ContainerAwareInterface
{
    protected $container;

    public function handle(RequestInterface $request, ParametersInterface
↪$configuration, ContextInterface $context)
    {
        $myService = $this->container->get('my_service');
```

(continues on next page)

(continued from previous page)

```
        $data = $myService->doSomething();

        return $this->response->build(200, [], [
            'hello' => $data,
        ]);
    }

    public function setContainer(ContainerInterface $container)
    {
        $this->container = $container;
    }
}
```

This works only in case you use a PHP class as action. For normal PHP files and Javascript files it is not possible to access the DI container.

5.3 Testing

Fusio provides a complete Test-Setup for your API endpoints. For the test case we use an in-memory sqlite database which contains the schema defined in the `resources/migration` folder. In the `Fixture.php` class it is also possible to define fixture data which is inserted for every test case.

The idea is that each endpoint has a corresponding test case class which tests the GET, POST, PUT and DELETE method of the resource. Internally we can send an HTTP request to Fusio without the need to setup an HTTP server. This makes these tests very fast and efficient.

The Method `Fixture::getPhpUnitDataSet` returns the data set which is inserted for every test case. There we insert a fixed access token with the fitting rights so that we can call our protected API endpoints.

You can execute those tests inside the Fusio directory root with a simple phpunit command (because of the available `phpunit.xml` configuration):

```
phpunit
```

5.3.1 Development

Every test case should extend from the `ApiTestCase` class. The test case contains a test method for every HTTP method i.e. `testGet`, `testPost`, etc. Every test makes the appropriated call to the API endpoint. Then we assert the response body and if needed also the headers (for larger response bodies it is recommended to move the expected JSON payload to an external file which is then included i.e. through `file_get_contents`). Through this way we can simply assure that our API works as expected. The following shows a simple API test case from the example todo entity API endpoint:

```
<?php

class EntityTest extends ApiTestCase
{
    public function testGet()
    {
        $response = $this->sendRequest('/todo/4', 'GET', [
            'User-Agent'    => 'Fusio TestCase',
        ]);
```

(continues on next page)

(continued from previous page)

```
        $actual = (string) $response->getBody();
        $actual = preg_replace('/\d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2}/', '0000-00-00␣
↪00:00:00', $actual);
        $expect = <<<'JSON'
{
    "id": "4",
    "status": "1",
    "title": "Task 4",
    "insertDate": "0000-00-00 00:00:00"
}
JSON;

        $this->assertEquals(200, $response->getStatusCode(), $actual);
        $this->assertJsonStringEqualsJsonString($expect, $actual, $actual);
    }

    public function testPost()
    {
        $response = $this->sendRequest('/todo/4', 'POST', [
            'User-Agent'    => 'Fusio TestCase',
        ]);

        $actual = (string) $response->getBody();
        $expect = <<<'JSON'
{
    "success": false,
    "title": "Internal Server Error",
    "message": "Given request method is not supported"
}
JSON;

        $this->assertEquals(405, $response->getStatusCode(), $actual);
        $this->assertJsonStringEqualsJsonString($expect, $actual, $actual);
    }

    public function testPut()
    {
        $response = $this->sendRequest('/todo/4', 'PUT', [
            'User-Agent'    => 'Fusio TestCase',
        ]);

        $actual = (string) $response->getBody();
        $expect = <<<'JSON'
{
    "success": false,
    "title": "Internal Server Error",
    "message": "Given request method is not supported"
}
JSON;

        $this->assertEquals(405, $response->getStatusCode(), $actual);
        $this->assertJsonStringEqualsJsonString($expect, $actual, $actual);
    }

    public function testDelete()
    {
        $response = $this->sendRequest('/todo/4', 'DELETE', [
```

(continues on next page)

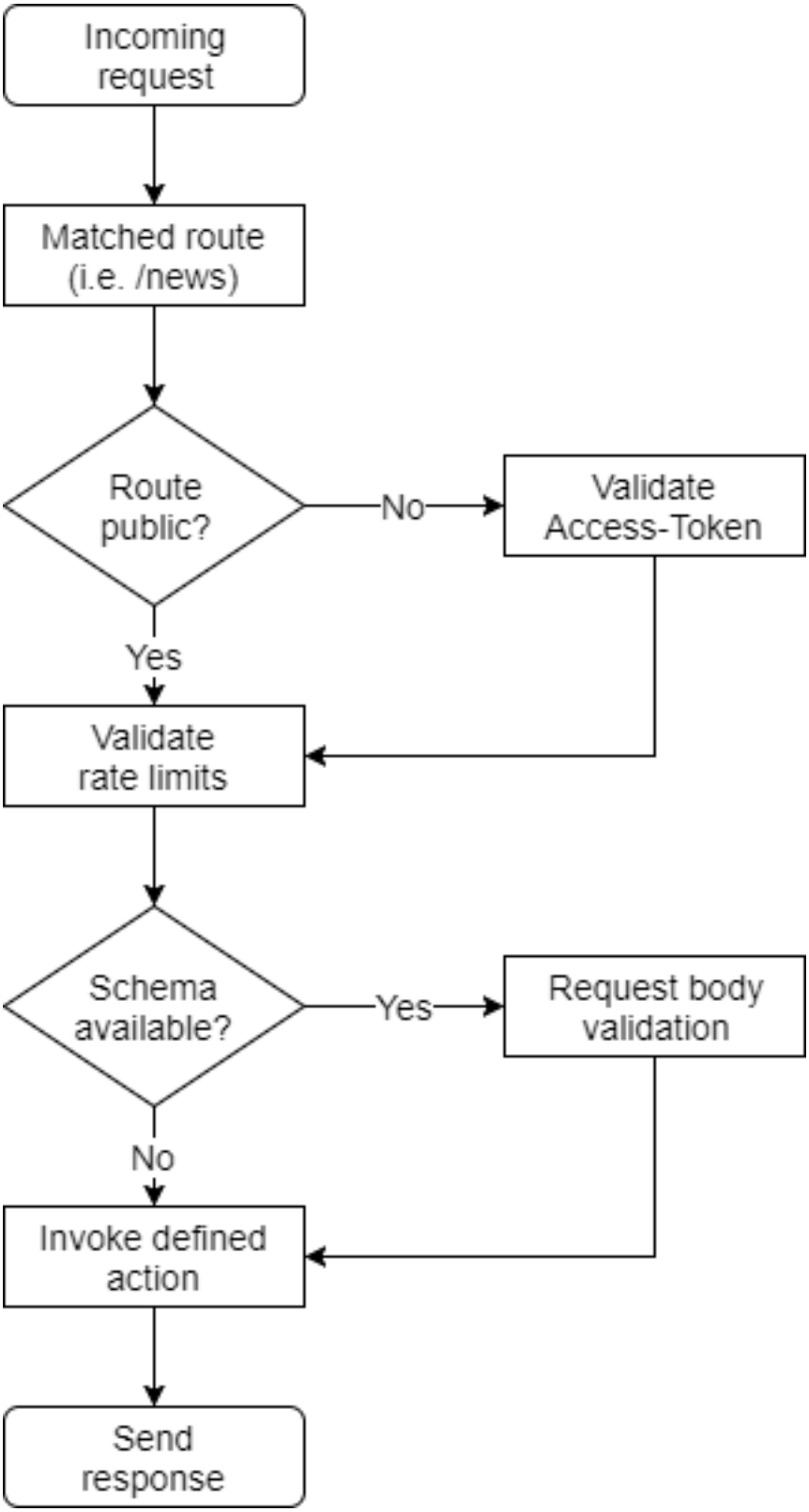
(continued from previous page)

```
            'User-Agent'    => 'Fusio TestCase',
            'Authorization' => 'Bearer␣
↪da250526d583edabca8ac2f99e37ee39aa02a3c076c0edc6929095e20ca18dcf'
        ]);

        $actual = (string) $response->getBody();
        $expect = <<<'JSON'
{
    "success": true,
    "message": "Delete successful"
}
JSON;

        $this->assertEquals(200, $response->getStatusCode(), $actual);
        $this->assertJsonStringEqualsJsonString($expect, $actual, $actual);

        /** @var \Doctrine\DBAL\Connection $connection */
        $connection = Environment::getService('connector')->getConnection('Default-
↪Connection');
        $actual = $connection->fetchAssoc('SELECT id, status, title FROM app_todo␣
↪WHERE id = 4');
        $expect = [
            'id' => 4,
            'status' => 0,
            'title' => 'Task 4',
        ];

        $this->assertEquals($expect, $actual);
    }

    public function testDeleteWithoutAuthorization()
    {
        $response = $this->sendRequest('/todo/4', 'DELETE', [
            'User-Agent'    => 'Fusio TestCase',
        ]);

        $actual = (string) $response->getBody();
        $expect = <<<'JSON'
{
    "success": false,
    "title": "Internal Server Error",
    "message": "Missing authorization header"
}
JSON;

        $this->assertEquals(401, $response->getStatusCode(), $actual);
        $this->assertJsonStringEqualsJsonString($expect, $actual, $actual);
    }
}
```

CHAPTER 6

Request lifecycle

To give you a first overview, every request which arrives at Fusio goes through the following lifecycle:



Fusio tries to assign the incoming request to a fitting route. The route contains all schema information about the incoming request and outgoing responses. Those schemas are also used at the documentation which is automatically available. If a request schema was provided the incoming request body gets validated after this schema. In case

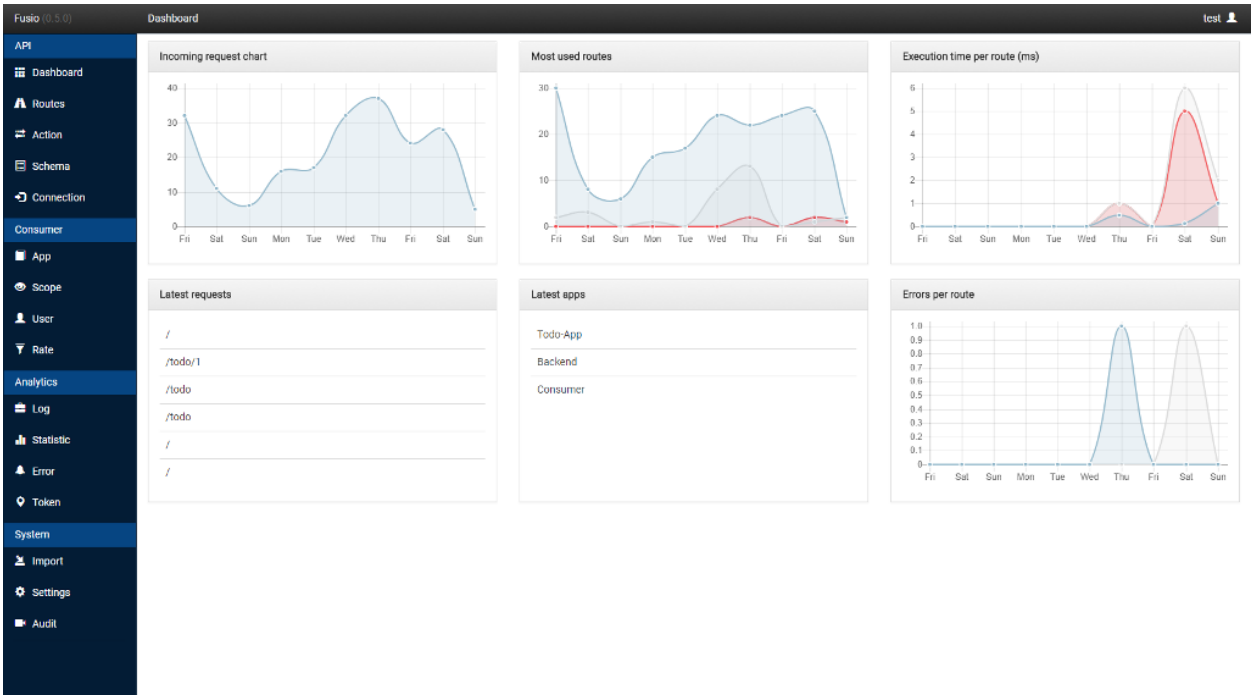
everything is ok the action which is assigned to the route gets executed.

An action represents the code which handles an incoming request and produces a response. Each action can use connections to accomplish this task. A connection uses a library which helps to work with a remote service. I.e. the SQL connection uses the Doctrine DBAL library to work with a database (it returns a `Doctrine\DBAL\Connection` instance). A connection always returns a fully configured object so you never have to deal with any credentials in an action. Besides that there are already many different actions available which you can use i.e. to create an API based on a database table.

With Fusio we want to remove as many layers as possible so that you can work in your action directly with a specific library. Because of this Fusio has no model or entity system like many other frameworks, instead we recommend to write plain SQL in case you work with a relational database. We think that building API endpoints based on models/entities limits the way how you would design a response. You only need to describe the request and response in the JSON schema format. This schema is then the contract of your API endpoint, how you produce this response technically is secondary. Fusio provides the mentioned connections, which help you to create complete customized responses based on complicated SQL queries, message queue inserts or multiple remote HTTP calls.

CHAPTER 7

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:

7.1 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 `src/` folder for more information.

7.2 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 information about the OAuth2 flow.

7.2.1 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.

7.2.2 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.

7.2.3 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.

7.3 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

7.4 Connection

A connection enables Fusio to connect to other remote sources. This can be i.e. a database or message queue server.

7.5 Import

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

7.6 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.

7.7 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.

7.7.1 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

7.7.2 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

7.7.3 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.

7.8 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 $ref key and the schema protocol i.e. schema:///[schema-name]. More detailed information about the json schema format at the RFC.

7.8.1 Example

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

7.9 Scope

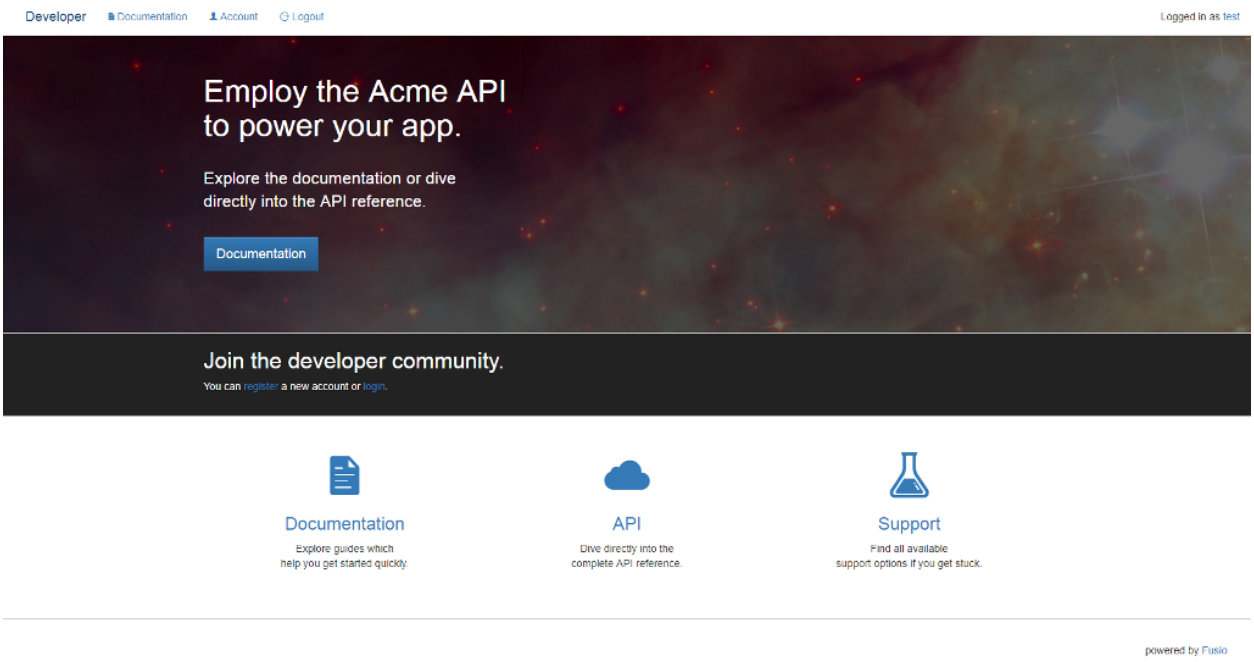
A scope describes the right to access specific routes and request methods. Each user account has assigned a set of allowed scopes. If a user creates an app he can only assign the scopes which are available for him.

7.10 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.

CHAPTER 8

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.

8.1 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.

8.2 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.

8.3 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.

CHAPTER 9

API

If you want to access or use the internal REST API of Fusio you can take a look at our internal API documentation. This documentation contains all available REST API endpoints.

CHAPTER 10

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. Our website lists every available composer package which has the `fusio-adapter` keyword defined in the `composer.json` file.

The adapter needs to require the `fusio/engine` package and must have an adapter class which implements the `Fusio\Engine\AdapterInterface` interface. This interface has a method `getDefinition` which returns an absolute path to a `adapter.json` definition file. 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. There is also a complete JsonSchema which describes this format.

```
{
    "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,
                    "request": "Adapter-Schema",
                    "responses": {
                        "200": "Passthru"
                    },
                    "action": "Void-Action"
                }
            }
```

(continues on next page)

(continued from previous page)

```
        }]
    }],
    "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
```