PSWebApp Documentation

Release 1.0

Malcolm Mackay

Sep 29, 2021

CONTENTS

See also cuplcodec and cuplfrontend. `pdf`

CHAPTER

ONE

API SPECIFICATIONS

1.1 Admin API Specification

Latest yaml AdminAPI

1.2 Consumer API Specification

Latest yaml ConsumerAPI

CHAPTER

TWO

DEVELOP LOCALLY

wsbackend has external dependencies such as a database and an identity provider. It is designed to be deployed to a cloud provider. Nonetheless it is easy to test this application locally without an internet connection.

2.1 Add / Edit an API endpoint

2.1.1 Update Specification

The API specification should be updated first. These are contained in files named `api.yaml`, which conform to the Swagger OpenAPI standard. It is sometimes easier to edit these files by copying them into and out of the Swagger web application.

2.1.2 Write a Test that Fails

Tavern is used for API testing. Simple tests are defined within yaml files that reside in the tests directory.

The script conftest.py contains pytest fixtures. These obtain access tokens or read environment variables on behalf of test scripts.

Executing `py.test` will run all tests against a live wsbackend server. Its address is set by environment variables:

- WSB_PROTOCOL
- WSB_HOST
- WSB_PORT

The newly developed application must be installed and made to serve pages at this address first! To automate this process I have created a Docker Compose file `docker-compose.test.yml` for quickly running tests locally.

First build all images with:

```
docker-compose -f docker-compose.test.yml build
```

This will take a few minutes the first time, but afterwards the docker layers will be cached so it should copy your new files in within seconds.

Next run tests with:

```
docker-compose -f docker-compose.test.yml up
```

2.1.3 Implement feature. Test until success

Push to GitHub. Tests will run nodejs.yml. A docker image is tested and built on every pull request. Readthedocs will execute on webookhook. If tagging the build then package is deployed to pypi. Build docker image.

2.2 Write a Frontend Application

When writing frontend applications you may want to deploy wsbackend locally. To do this I have created `docker-compose.yml`.

CHAPTER

THREE

WSBACKEND

3.1 Installation

This tutorial assumes you have Git and Python 3 installed.

3.1.1 Install Prerequisites

One of the dependencies (Postgres library Psycopg2) should be built from source.

GCC Compiler

To install GCC on Debian, open a terminal window and type:

```
$ sudo apt install build-essential
```

To check gcc is installed, type:

```
$ gcc --version
```

Python Header Files

To install the Python 3 header files type:

```
$ sudo apt-get install python3-dev
```

Libpq-dev

This To install libbq-dev type:

```
$ sudo apt-get install libpq-dev
```

3.1.2 Clone the Repo

First you will need to download wsbackend by cloning its Git repository. Open a terminal window, browse to a folder of your choice and type:

```
$ git clone https://github.com/websensor/wsbackend.git
```

The GitHub repository will be cloned to a folder named `wsbackend`. Open this by typing:

```
$ cd wsbackend
```

3.1.3 Create a Virtual Environment

We will use virtualbox to create a virtual environment. This will isolate our wsbackend installation from the rest of the system.

First install virtualenv:

```
$ sudo pip3 install virtualenv
```

Next create a virtual environment named `wsbackend_env`:

```
$ virtualenv wsbackend_env
```

3.1.4 Activate the Virtual Environment

In order for `pip` to install dependencies into the virtual enviornment, you must activate it first with:

```
$ source wsbackend_env/bin/activate
```

3.1.5 Install Dependencies

wsbackend is dependent on a number of Python packages including flask. These are listed in requirements.txt.

To install all requirements, type:

```
$ pip3 install -r requirements.txt
```

3.1.6 Define Environment Variables

Auth0

3.1.7 Install Gunicorn

There are many options for serving this Flask application. In this tutorial we will use Gunicorn. It is the easiest to install:

```
$ pip3 install gunicorn
```

Next run the application with:

```
$ gunicorn main:app
```

3.2 Environment Variables

3.2.1 Host address

`WSB_PROTOCOL`

Default: `http://`

Protocol of wsbackend.

`WSB_HOST`

Default: `localhost`

Hostname of wsbackend.

`WSB_PORT`

Default: `5000`

Port of wsbackend.

3.2.2 Database

`DB_HOST`

Default: `localhost`

Host where the database is running.

`DB_PORT`

Default: `5432`

Port number to connect to the database.

`DB_USER`

Default: `postgres`

Database user name.

`DB_PASS`

Database password.

No default for security reasons. This variable must be set before runtime.

3.2.3 Consumer API

`API_ISSUER`

Default: `http://localhost:3000`

API issuer claim. Must correspond to the issuer of the API access token.

`API_AUDIENCE`

Default: `mock_api_audience`

Access tokens signed by the IdP must contain this audience claim. Without it, the consumer API will reject the token and deny access to an endpoint that requires authorization.

Defaults to a value used by the mock IdP.

3.2.4 Admin API

`ADMINAPI_AUDIENCE`

Default: `default_adminapi_audience`

Access tokens issued using the client credentials flow will contain this audience claim.

`ADMINAPI_CLIENTID`

Default: `default_adminapi_clientid`

A token request using the client credentials flow must contain this `client_id`. This is a shared secret between the sender and `wsbackend`.

`ADMINAPI_CLIENTSECRET`

A token request using the client credentials flow must contain this `client_secret`. This is a shared secret between the sender and `wsbackend`.

No default for security reasons. This variable must be set before runtime.

3.3 Reference

3.3.1 API Layer

3.3.2 Service Layer

3.3.3 Model Layer

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search