Smart Pots

Release 1.0.0

Blahovici Andrei, Dumitrescu Delia, Ganea Antonio, Preda Mihai,

Feb 01, 2022

CONTENTS:

Smart Pots is an IoT project implementing remote care for a potted plant, through sensors and robotics which monitor its state.

Please refer to *Installation* to get started with using Smart Pots.

Once you've *installed Smart Pots*, we recommend reading the *tutorial*.

For a complete list of features, see *features*.

The code can be found on GitHub at Shest-Programmistov/Smart-Pots. Data has been taken from IoTsec/Room-Climate-Datasets.

CHAPTER

ONE

TOOLS USED

- For the HTTP connection, we are using the Flask library.
- For the MQTT connection, the Flask-MQTT extension is used. Also, for mocking the MQTT subscriber, paho-mqtt is used.
- The database is built using SQLite - a fully open-source RDBMS.

1.1 Installation

1.1.1 Prerequisites

Precondition python3 and pip3 are installed. Also, you should have an MQTT Broker installed.

Mosquitto MQTT Broker Installation

For installing Mosquitto MQTT Broker, go to their official website and download and install the Mosquitto Broker for your OS.

For Ubuntu/Debian:

Run:

```
sudo apt-get install mosquitto
```

To check whether the service is running or not and to start it, run:

```
sudo systemctl status mosquitto # Checking if the service is running
sudo systemctl start mosquitto # Start the service
```

For Mac:

Install Mosquitto on Mac OS using Homebrew:

```
brew install mosquitto
```

1.1.2 Installation

1. Install virtualenv if not already installed:

On Linux, run:

```
sudo pip3 install virtualenv
```

2. Create a new virtual environment:

```
cd Smart-Pots/
virtualenv .venv
```

Note: Use *.venv* or any *$NAME* for your virtualenv.

3. Activate the environment:

```
source .venv/bin/activate
```

and you should see your *.venv* activated

```
(.venv) ~/Smart-Pots$
```

Note: To deactivate the environment, simply use *deactivate*.

4. Install the required libraries:

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

5. To switch Flask to the development environment and enable debug mode, set *FLASK_ENV*:

```
export FLASK_ENV=development
```

6. Initialize (or reinitialize) database:

```
flask init-db
```

1.2 Tutorial

First, refer to *Installation* for setting up Smart Pots. After having installed successfully the requirements, follow the next steps:

1.2.1 Running on Linux

1. **Start the MQTT Broker service**. The Broker represents an intermediary entity that enables the MQTT clients to communicate.

If Mosquitto is used, run:

```
sudo service mosquitto start
```

To test if it is running use the *netstat –at* command. You should see the Mosquitto broker running on port 1883.

To stop the service, use *sudo service mosquitto stop*.

2. Running the application. The proper way to run the application is:

```
python3 app.py
```

This way, we make sure that SocketIO is used, as Flask is wrapped in SocketIO.

Note: To only run the Flask app (no MQTT communication), just use:

```
flask run
```

[Optional] 3. Run the MQTT subscriber to check that data is successfully received.

```
python3 mqtt_comms_sub.py
```

1.2.2 Testing

To run the tests, simply execute:

```
pytest
```

To measure the code coverage, run:

```
coverage run -m pytest
```

and then use coverage report to report on the results:

```
coverage report -m
```

For a nicer presentation, use *coverage html* to get annotated HTML listings detailing missed lines.

1.2.3 Developer Tools

OpenAPI

We used the OpenAPI Initiative (OAI) to specify what our API can do.

The Swagger API can be accessed at:

```
http://127.0.0.1:5000/api/docs
```

AsyncAPI

The AsyncAPI Specification is a comprehensive specification language for describing asynchronous messaging APIs.

If AsyncAPI Generator is not installed, you can install it by running:

```
npm install -g @asyncapi/generator
```

Then, run:

```
ag water.yml @asyncapi/html-template -o output
```

1.3 Features

1.3.1 Authentication

Authentication is integrated so that the Smart Pot functionalities are available only with logging-in in advance.

`http_routes.auth.login()`
Logs in a user. — parameters:

- in: body name: body schema:

  required:

  – username

  – password

  properties:

  username: type: string description: the login name for the user

  password: type: string description: the password for the user

responses:

200: description: user logged in succesfully.

403: description: there is no user with that username and password.

422: description: required parameters not supplied.

`http_routes.auth.logout()`
Logs out the current user. — responses:

200: description: user logged out succesfully.

403: description: user is not authenticated.

`http_routes.auth.register()`
Registers a new user. — parameters:

- in: body name: body schema:

  required:

  – username

  – password

  properties:

username: type: string description: the login name for the new user

password: type: string description: the password for the new user

responses:

200: description: user registered succesfully.

403: description: there is already an user with that username

422: description: required parameters not supplied.

1.3.2 Plant Characteristics Setting

Manages setting the plant species, as the needed amount of water differs based on the plant.

`http_routes.characteristics.set()`
Sets the characteristics of the plant. — parameters:

- in: body name: body schema:

  required:

  – ideal_humidity

  – ideal_temperature

  properties:

  ideal_humidity: type: number description: the ideal humidity for the plant

  ideal_temperature: type: number description: the ideal temperature for the plant

responses:

200: description: everything went fine.

403: description: user is not authenticated.

422: description: required parameters not supplied.

1.3.3 Temperature Monitoring

Temperature endpoint

`http_routes.temperature.set()`
Sets the temperature level. — parameters:

- in: body name: body schema:

  required:

  – degrees

  properties:

  degrees: type: number description: the number of degrees to set to

responses:

200: description: everything went fine.

403: description: user is not authenticated.

422: description: degrees not supplied.

1.3.4 Humidity Monitoring

Humidity endpoint

`http_routes.humidity.set()`
Sets the humidity level. — parameters:

- in: body name: body schema:

  required:

  – value

  properties:

  value: type: number description: the humidity level

responses:

200: description: everything went fine.

403: description: user is not authenticated.

422: description: value not supplied.

1.3.5 Manual Watering

Smart Pots also provides a manual watering functionality.

Endpoint force_water - when called, proceeds to water the plant with a specified value/ amount of water.

`http_routes.force_water.force_water()`
Waters the plant immediately. — parameters:

- in: body name: body schema:

  required:

  – value

  properties:

  value: type: number description: the quantity of water

responses:

200: description: everything went fine.

403: description: user is not authenticated.

422: description: value not supplied.

1.3.6 Watering Statistics

Endpoint returning plot showing the watering history.

http_routes.plot.**plot**()
Plots the water quantities over the last week divided by hours. — responses:

200: description: everything went fine.

403: description: user is not authenticated.

http_routes.plot.**plot_humidity**()
Plots the humidity over the last week. — responses:

200: description: everything went fine.

403: description: user is not authenticated.

http_routes.plot.**plot_temperature**()
Plots the temperature over the last week. — responses:

200: description: everything went fine.

403: description: user is not authenticated.

1.3.7 Automatic Watering

The smart pot acts as the "publisher" - constantly broadcasting the amount of water that should be provided to the plant.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX