pymipago

Release 1.1.dev0

Mikel Larreategi

Oct 04, 2021

Contents:

CHAPTER 1

pymipago

Python package to make payment requests with Basque Government's payment service

- Free software: GNU General Public License v3
- Documentation: https://pymipago.readthedocs.io.

1.1 Features

This package allows to use the Basque Governent's Payment Service, a platform that allows Basque public institutions to receive payments from the citizens.

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

The development of this package was funded by AMETX Erakunde Autonomoa

CHAPTER 2

Installation

2.1 Stable release

To install pymipago, run this command in your terminal:

```
$ pip install pymipago
```

This is the preferred method to install pymipago, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for pymipago can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/codesyntax/pymipago
```

Or download the tarball:

```
$ curl  -OL https://github.com/codesyntax/pymipago/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage of the library

There is one method to make payment requests using the Basque Government's payment service:

```
from pymipago import make_payment_request
```

This method creates an XML file and creates a payment request on the Government platform in order to have the basis to be shown to the end user.

According to the payment platform specs, after the registration, an HTML file is created which must be shown to the user. The contents of this HTML file are returned when the method is called. The method also returns the payment_code that the user should save for further checks.

This HTML file has an "auto-refresh" feature which allows to redirect the user to the payment platform, where all the data of the payment is already entered.

There, the enduser only has to select the bank of his choice to complete the payment.

After completing the payment the user will be redirected to the *return_url*.

See the documentation for more information about the parameters

This method takes some parameters, the explanation of them is the following:

- **cpr**: right now the only allowed value is '9052180' which allows the use of the so called "Cuaderno 60" and "Formato 521"
- **sender**: is a 6 digit sender code. This code will be assigned by the Government platform in which the sender must be registered prior to the use of this library.
- **format**: right now the only allowed value is '521' which allows the use of the so called "Cuaderno 60" and "Formato 521"
- **suffix**: is a 3 digit code. This code must be created by the sender in the Government platform.
- **reference_number**: is a 10 digit code. This code is created by the sender to identify the payments.
- **payment_limit_date**: is a datetime.date object with the date before which the payment must be completed. It must be a date in the future.

- **quantity**: how much must the user pay. It must be a string value in euro cents. For example: if 50 € must be payed, the value must be '5000'. If 11,50 € must be payed the value must be '1150'
- **language**: 2 letter code of the language in which the payment screen should be shown. Government platform only allows to select 'eu', 'es' and 'en'. If any other value is used, the screen is presented in 'es'
- **return_url**: a valid URL where the user will be redirected after the payment is completed.
- **payment_modes: a list of 2 letter codes representing the payment mode. There are 2 payment modes enabled on the Gove**
  - '01': offline payment: the user has to download a PDF file and go to a bank to complete the payment
  - '02': online payment: the user is presented a list of online bank platforms to complete the payment
- **test_environment**: (default: False) a boolean to use the testing environment of the Payment Service.
- **extra: (default: {}): a dict to override default values of the payment service configuration. Currently supported values are**
  - 'message1': format: {'eu': 'XX', 'es': 'XX'}: basque and spanish texts to override the footer value of the payment document in PDF format
  - 'message2': format: {'eu': 'XX', 'es': 'XX'}: basque and spanish texts to override the first legal text of the payment document in PDF format
  - 'message3': format: {'eu': 'XX', 'es': 'XX'}: basque and spanish texts to override the second legal text of the payment document in PDF format
  - 'message4': format: {'eu': 'XX', 'es': 'XX'}: basque and spanish texts to override the header text of the payment document in PDF format
  - 'message_payment_title': format: {'eu': 'XX', 'es': 'XX'}: basque and spanish text to override the name of the payment .
  - 'message_payment_description': format: {'eu': 'XX', 'es': 'XX'}: basque and spanish text to show the concept of the payment.
  - 'citizen_name': text to show citizen's citizen_name in the payment document.
  - 'citizen_surname_1': text to show citizen's citizen_surname_1 in the payment document.
  - 'citizen_surname_2': text to show citizen's citizen_surname_2 in the payment document.
  - 'citizen_nif': text to show citizen's citizen_nif in the payment document.
  - 'citizen_address': text to show citizen's citizen_address in the payment document.
  - 'citizen_postal_code': text to show citizen's citizen_postal_code in the payment document.
  - 'citizen_territory': text to show citizen's citizen_territory in the payment document.
  - 'citizen_country': text to show citizen's citizen_country in the payment document.
  - 'citizen_phone': text to show citizen's citizen_phone in the payment document.
  - 'citizen_email': text to show citizen's citizen_email in the payment document.
  - 'logo_1_url': url of the 1st logo shown in the payment document. You need to add this logo previously in the Payment Service. Check with them for further assistance.
  - 'logo_2_url': url of the 2nd logo shown in the payment document. You need to add this logo previously in the Payment Service. Check with them for further assistance.
  - 'pdf_xslt_url': url of the XSLT template that will be used to render several templates. Check with the Payment Service for further assistance.

CHAPTER 4

pymipago

4.1 pymipago package

4.1.1 Submodules

4.1.2 pymipago.constants module

includes some constants used by the library

4.1.3 pymipago.exceptions module

exception pymipago.exceptions.**InvalidCPRValue**
Bases: exceptions.Exception

Raised when the used CPR value is not valid.

exception pymipago.exceptions.**InvalidFormatValue**
Bases: exceptions.Exception

Raised when the used Format value is not valid.

exception pymipago.exceptions.**InvalidReferenceNumber**
Bases: exceptions.Exception

Raised when the format of the reference_number is not valid.

exception pymipago.exceptions.**InvalidRegistration**
Bases: exceptions.Exception

Raised when the registration of the payment on the Government platform is invalid and has created an error

4.1.4 pymipago.utils module

util functions used by the main module

4.1.5 Module contents

pymipago.**make_payment_request**(*cpr, sender, format, suffix, reference_number, payment_limit_date, quantity, language, return_url, payment_modes=['01', '02'], test_environment=False, extra={}*)

This method creates an XML file and creates a payment request on the Government platform in order to have the basis to be shown to the end user.

According to the payment platform specs, after the registration, an HTML file is created which must be shown to the user. This HTML file has an "auto-refresh" feature which allows to redirect the user to the payment platform, where all the data of the payment is already entered.

There, the enduser only has to select the bank of his choice to complete the payment.

After completing the payment the user will be redirected to the *return_url*.

See the documentation for more information about the parameters

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/codesyntax/pymipago/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

pymipago could always use more documentation, whether as part of the official pymipago docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/codesyntax/pymipago/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *pymipago* for local development.

1. Fork the *pymipago* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pymipago.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pymipago
$ cd pymipago/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests:

```
$ flake8 pymipago tests
$ python setup.py test or py.test
```

   To get flake8 just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website. Travis-ci will be used to test your changes and the integration with master with several python versions.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6. Travis-CI will run the tests and report to the Pull Request.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_pymipago
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run *fullrelease* a command from zest.releaser which will help you creating a new version number, tagging it on git and bumpin the version number:

```
$ fullrelease
```

CHAPTER 6

Credits

6.1 Development Lead

- Mikel Larreategi <mlarreategi@codesyntax.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 1.1 (unreleased)

- Nothing changed yet.

7.2 1.0 (2020-01-21)

- Use new Travis CI [erral]
- Remove bumpversion as dependency [erral]

7.3 1.0b7 (2018-07-24)

- Force control digits [erral]
- Test against python 3.7 [erral]
- Update dependency versions [erral]

7.4 1.0b6 (2018-06-01)

- Fix control digit calculation [erral]
- Document new extra parameters for logos and XSLT template. [erral]

7.5 1.0b5 (2018-04-27)

- Add extra parameters to send logo urls [erral]

- Add an extra parameter to send the XSLT template [erral]

7.6 1.0b4 (2018-04-25)

- Add an extra parameter to register additional optional information into the payment service [erral]

7.7 1.0b3 (2018-04-20)

- Previous was an errored relase. [erral]

7.8 1.0b2 (2018-04-20)

- Add a parameter to use the testing environment of the Payment service. [erral]

7.9 1.0b1 (2018-04-18)

- Implementation of notebook 60 payments in short format (521) [erral]

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index