

flask-catalog

Portfolio project to demonstrate a Flask based implementation of a secure user model with registration, email confirmation, and Google oauth2 login, combined with an example catalog of items grouped in categories. Registered users can CRUD the catalog via a web front-end or via a REST API that adheres strictly to the [JSON API 1.0](#) specification. The REST API supports advanced searching and filtering. End 2 end scenarios are demonstrated via a python client written in a [Jupyter notebook](#).

Deployed on Heroku

The application is deployed on <https://flask-catalog.herokuapp.com/>, so please follow along and interact with the front end via a browser or interact with the back end via API calls.

Front End

Home: <https://flask-catalog.herokuapp.com/>

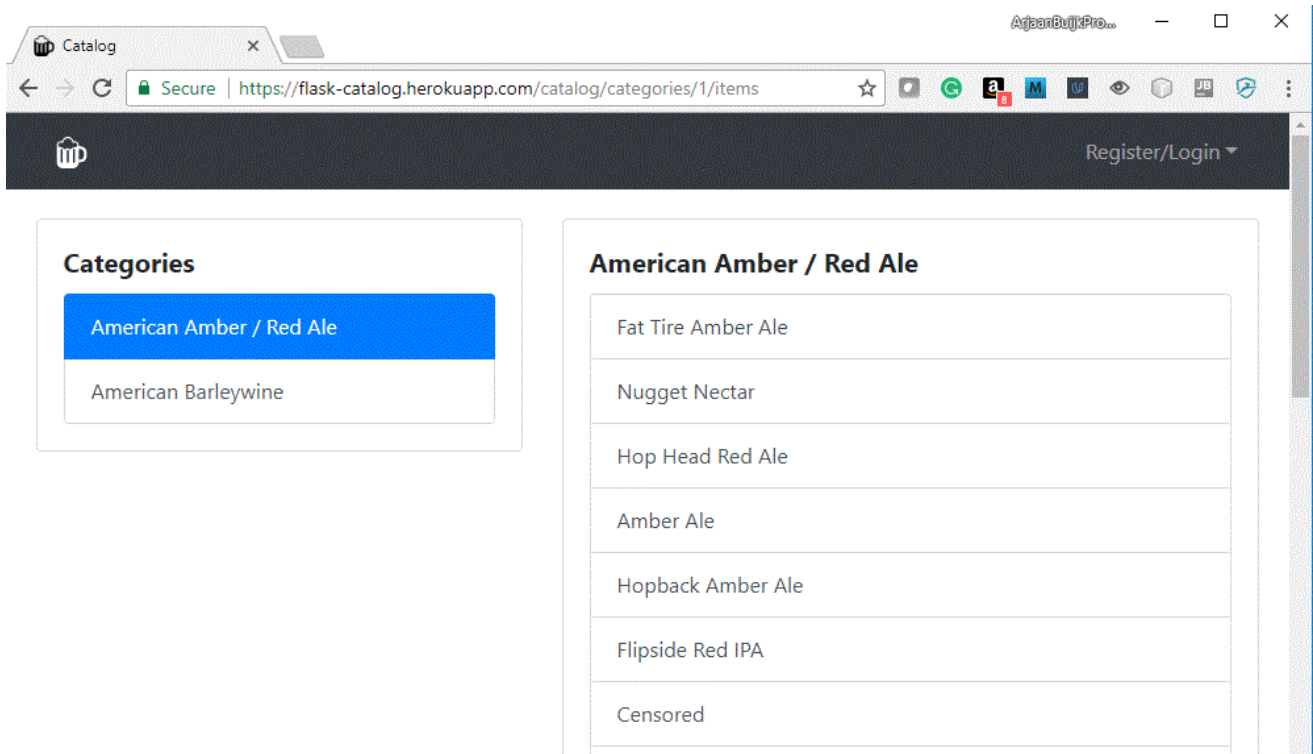


© Arjaan Buijk 2018

The home page shows a Jumbotron with some branding, three buttons (Visit, Register, Log in), credit to content providers and a copyright at the bottom.

Note the favicon icon in the browser tab, which shows a beer stein, since the catalog is populated with beers.

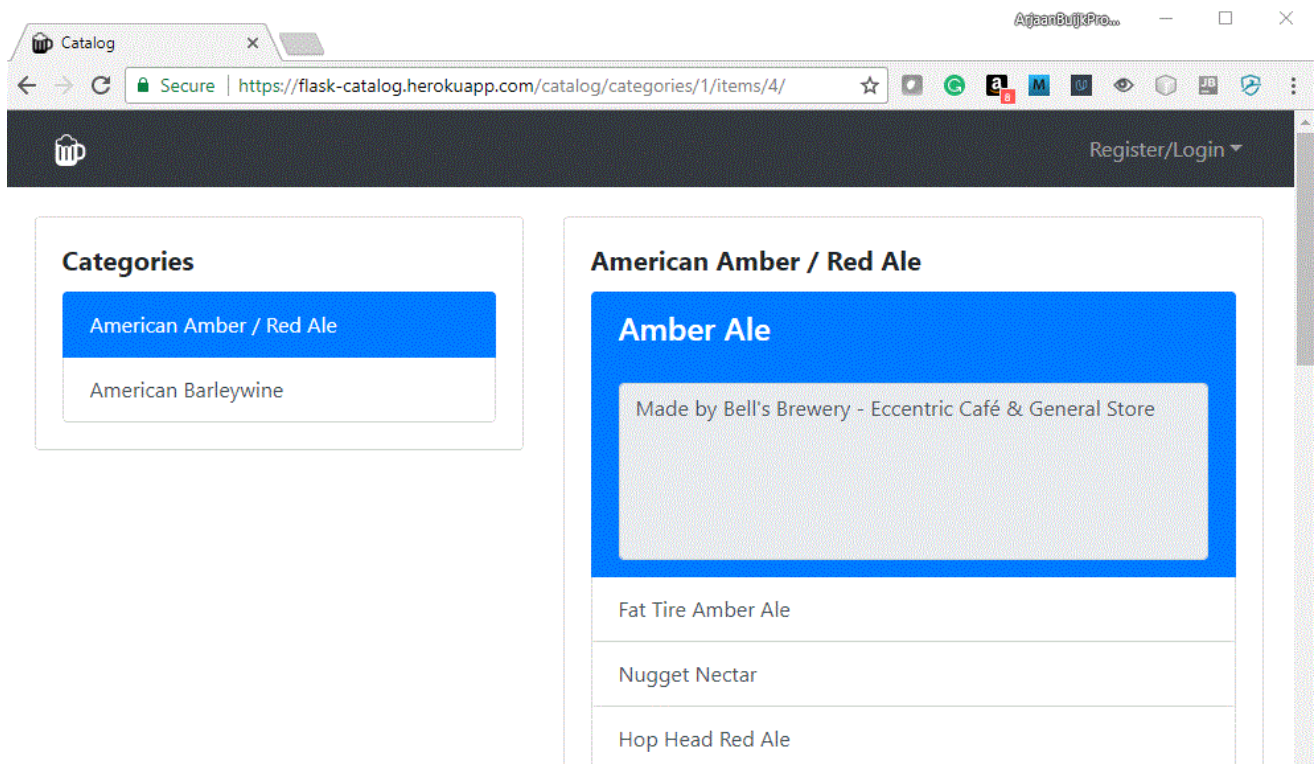
Main page - categories: <https://flask-catalog.herokuapp.com/catalog/categories/1/items>



When you click the Visit>> button on the home page, you are directed to the main page of the catalog, where you see the current content of the catalog.

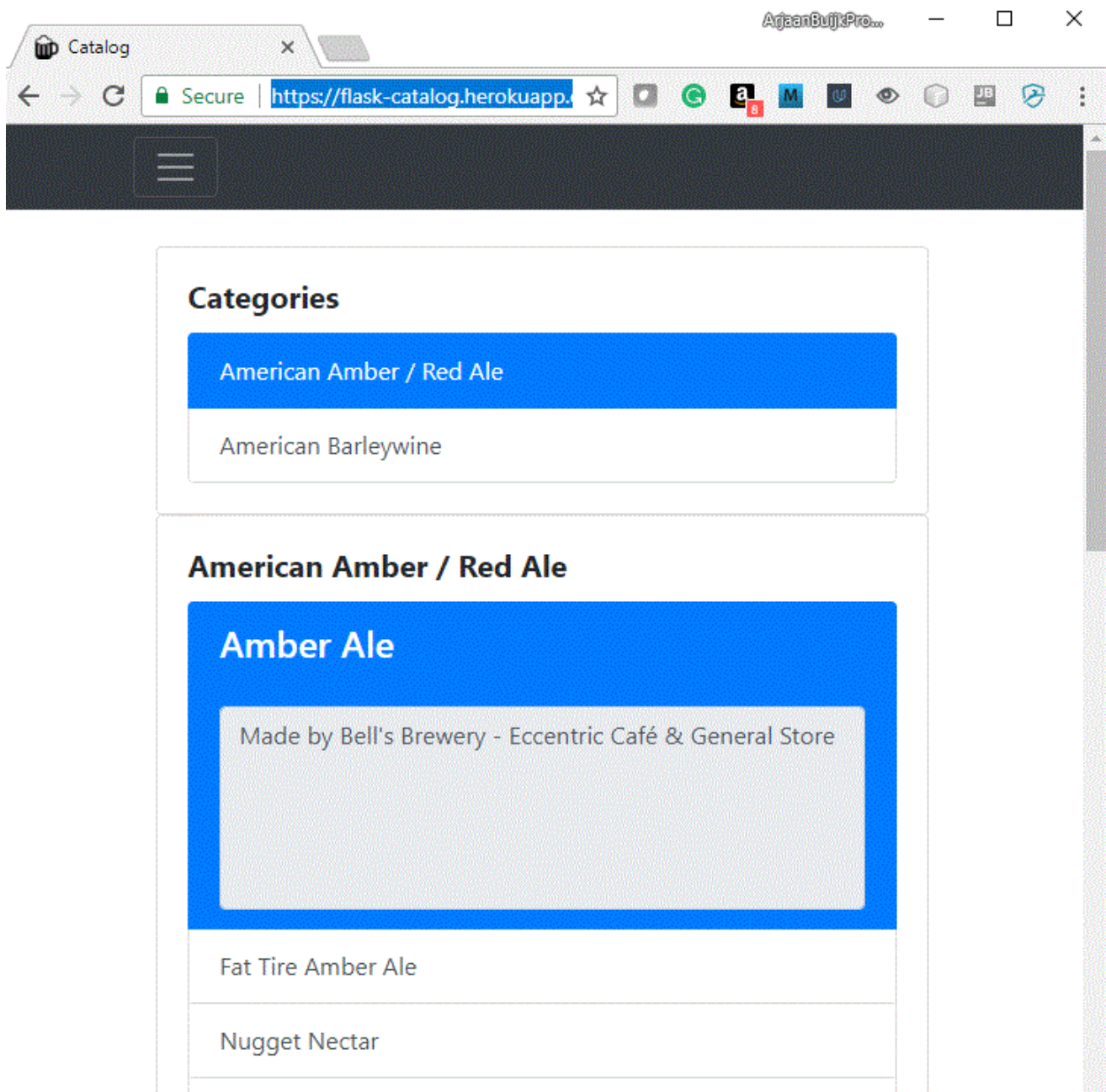
The Categories are listed on the left side of the page, while the beers (items) that belong to the selected category are listed on the right side.

Main page - items: <https://flask-catalog.herokuapp.com/catalog/categories/1/items/4/>



When you click on a beer in the list on the right side, a redirection takes place to the selected item in the selected category. The example image shows how the selected beer is always shown at the top of the list, with an expanded view where the description is given.

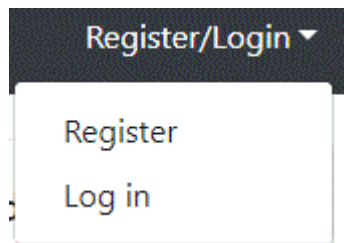
Navbar & Windows collapse: <https://flask-catalog.herokuapp.com/catalog/categories/1/items/4/>



When you change the browser window size, notice how the Navbar is nicely collapsing and how the lists of Categories and Items are re-organizing themselves vertically.

This pretty much completes what a visitor of the catalog can do without logging in.

Register/Login dropdown




When a user is not authenticated, the Navbar shows a dropdown on the right side that provides options for Register and Log in.

Login: <https://flask-catalog.herokuapp.com/login>

Log in

Secure | <https://flask-catalog.herokuapp.com/login>

Register/Login

 Log in

Email

Password


☐ Remember me

Log in

No account? [Register](#)

Forgot your password? [Reset password](#)

or sign in with this service

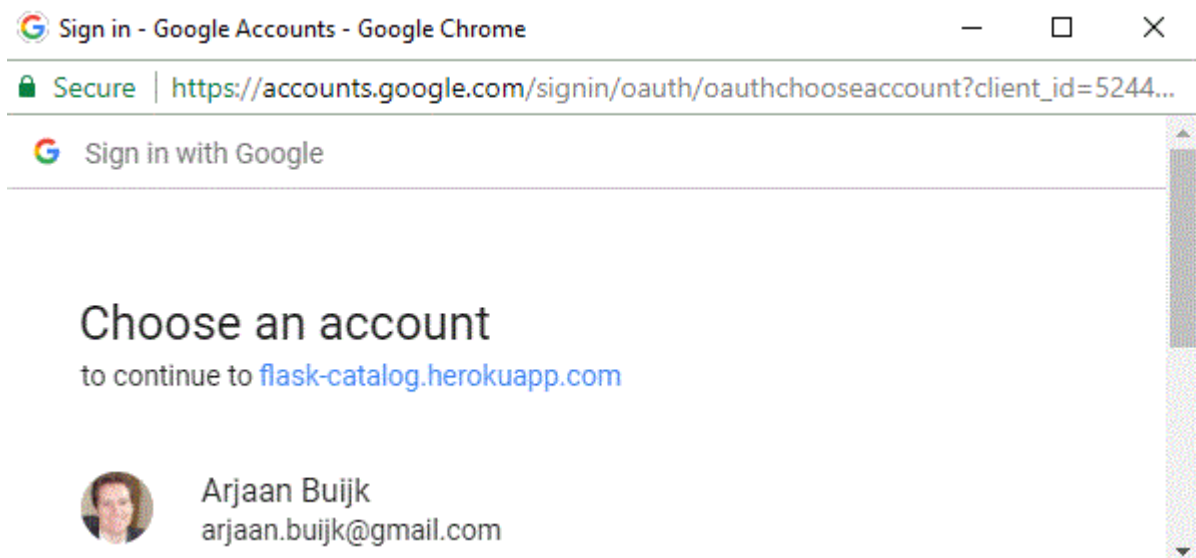
 Sign in

© Arjaan Buijk 2018

The login page can be reached from the home page or from the Navbar dropdown.

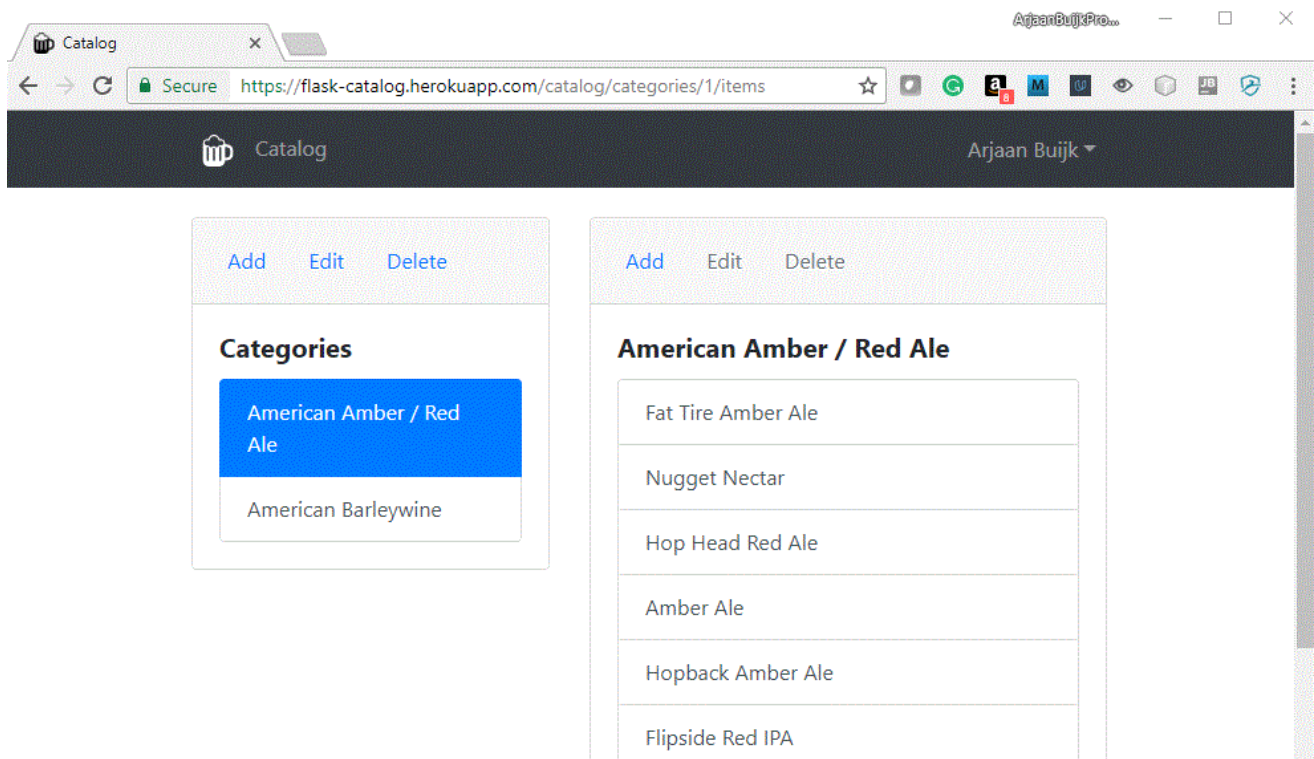
As the image shows, one can login with an email & password, but that requires you to first register an account. A more convenient approach is to sign in with the service listed at the bottom, which uses Google OAUTH2 to authenticate you and log you in. Please first login with your google account.

Google Login



When you click on the Google Sign in button, you will be asked to authenticate yourself with a gmail address. Upon successful authentication, the flask-catalog application will verify on the server side that all is indeed OK by making a call to the Google OAUTH2 API. If all is fine, your email will be registered as a confirmed user, and you will be logged in.

Main page - Logged In: <https://flask-catalog.herokuapp.com/catalog/categories/1/items>

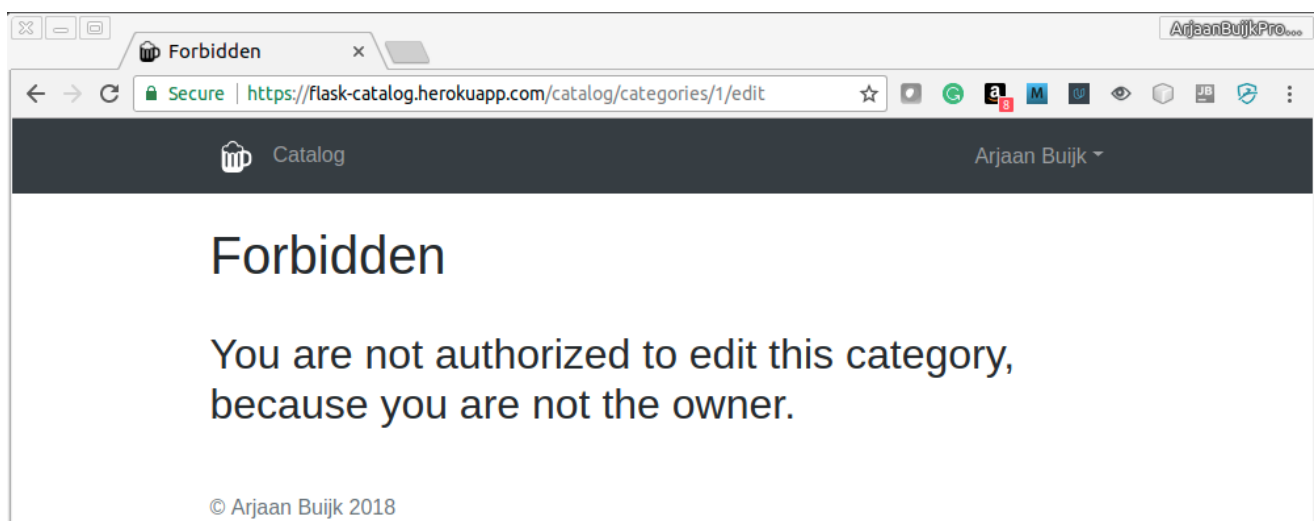


Once logged in, you will be redirected to the main page of the categories with the first category selected.

Note that above both the categories and the items lists you can find options to **Add, Edit or Delete**. The Edit and Delete options for the items are deactivated until you select an item. This is also indicated by a tooltip 'Please select an item first' when you hover the mouse above them.

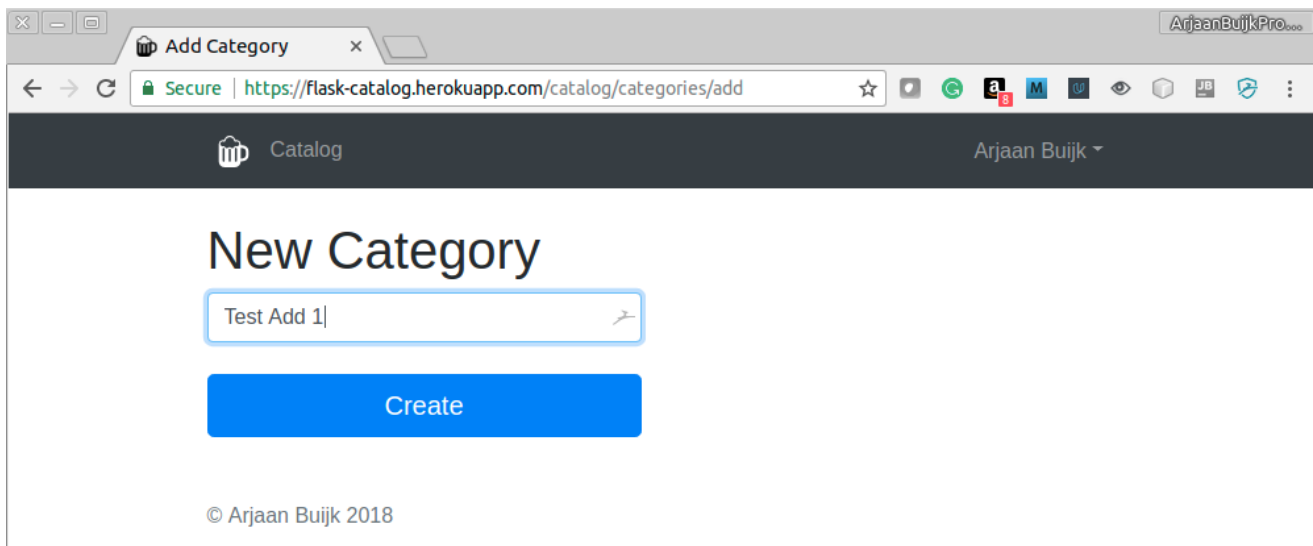
Also note that the Navbar has changed. A link is added to get back to this main page, and the dropdown now shows your name.

Edit Category that you do not own: <https://flask-catalog.herokuapp.com/catalog/categories/1/edit>



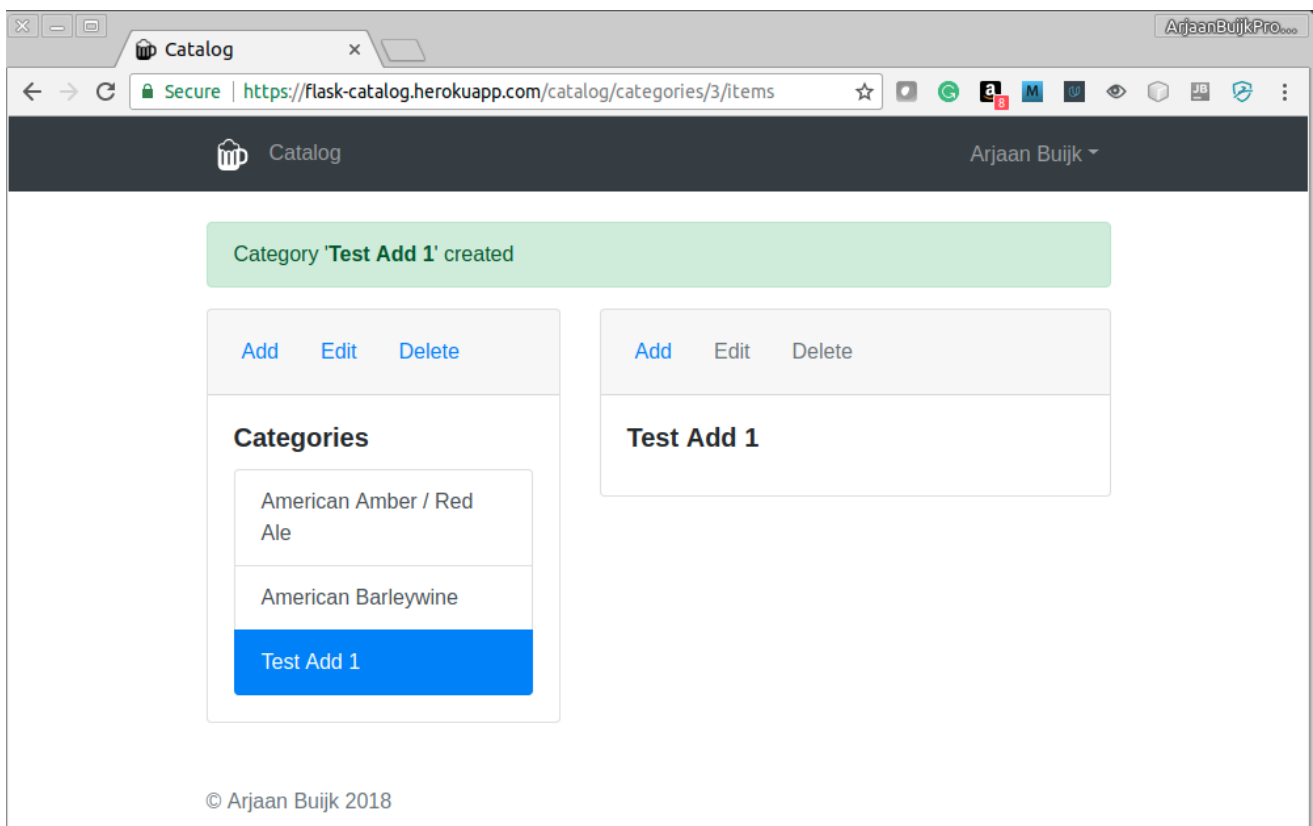
When you try to edit the first Category, you will be redirected to a custom 403.html page that looks like the image shown. This will happen for Edit & Delete actions on **categories** and **items** that you do not own.

Add Category: <https://flask-catalog.herokuapp.com/catalog/categories/add>



© Arjaan Buijk 2018

So, lets add a Category to our beer catalog. When clicking the add link, it brings up a page where you can fill out the name of the new category, and then click Create.



© Arjaan Buijk 2018

The new category is added to the database, and you will be redirected to a page that shows all the items in this new category, which off course is an empty list for now.

Note that a message is flashed back to you to confirm that the category was added.

Add Item: <https://flask-catalog.herokuapp.com/catalog/categories/3/items/add>

ArjaanBuijkPro

Add Item

Secure | <https://flask-catalog.herokuapp.com/catalog/categories/3/items/add>

Catalog Arjaan Buijk

New Item

Category: Test Add 1

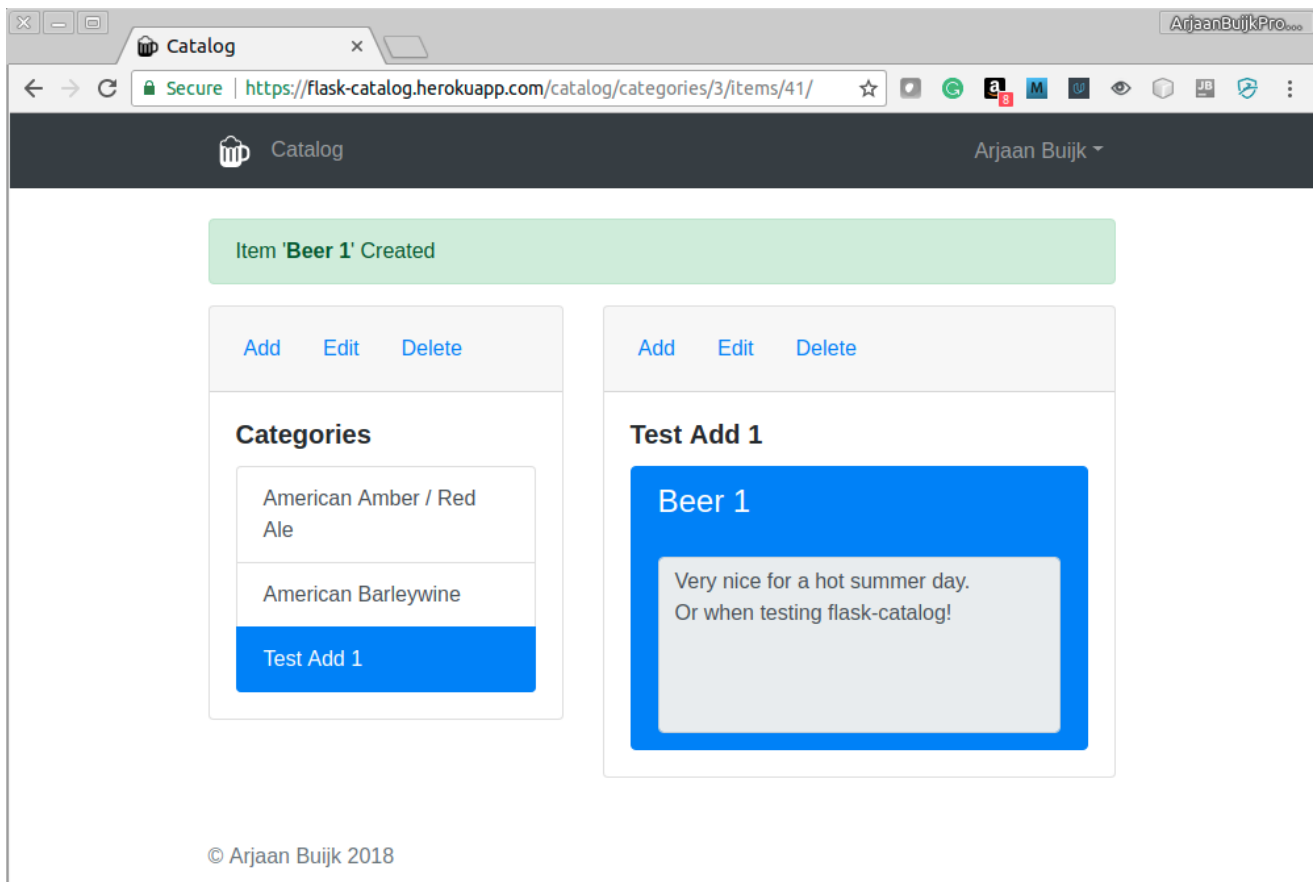
Beer 1

Very nice for a hot summer day.
Or when testing flask-catalog!

Create

© Arjaan Buijk 2018

Lets add some beers to our new category, by clicking on the Add link in the right column. This brings up a page where you can fill out the name of the new beer with a description, and then click Create.



The new beer is added to the database, and you will be redirected to a page that shows all the items in the category, with the newly selected beer selected and shown at the top of the page.

Note that a message is flashed back to you to confirm that the category was added.

Lets try this again:

ArjaanBuijkPro

Catalog

Secure | <https://flask-catalog.herokuapp.com/catalog/categories/3/items/43/>

Catalog Arjaan Buijk

Item 'Beer of the month' Created

Add Edit Delete

Categories

American Amber / Red Ale

American Barleywine

Test Add 1

Add Edit Delete

Test Add 1

Beer of the month

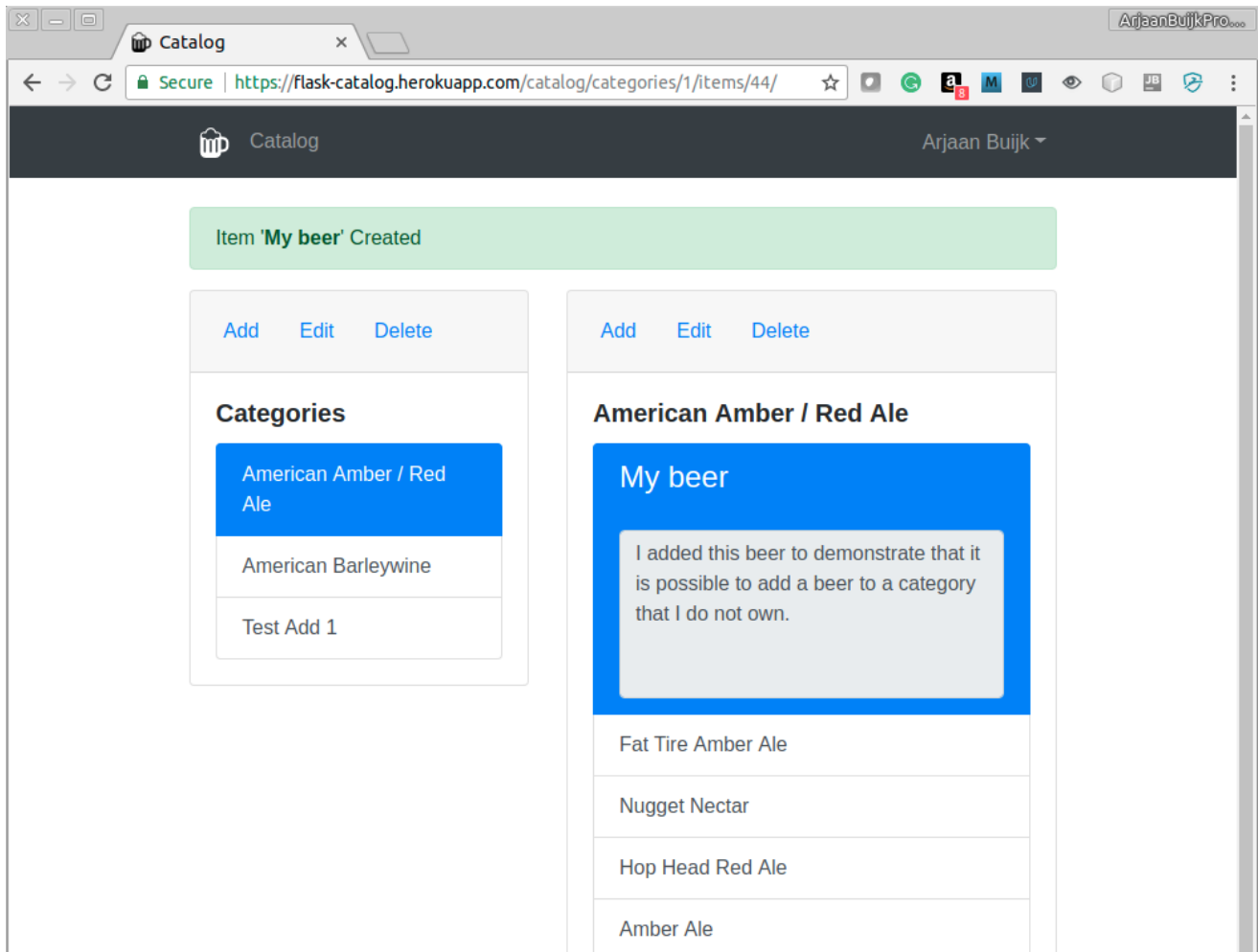
A favorite among programmers.
At a reasonable price!

Beer 1

Another nice beer

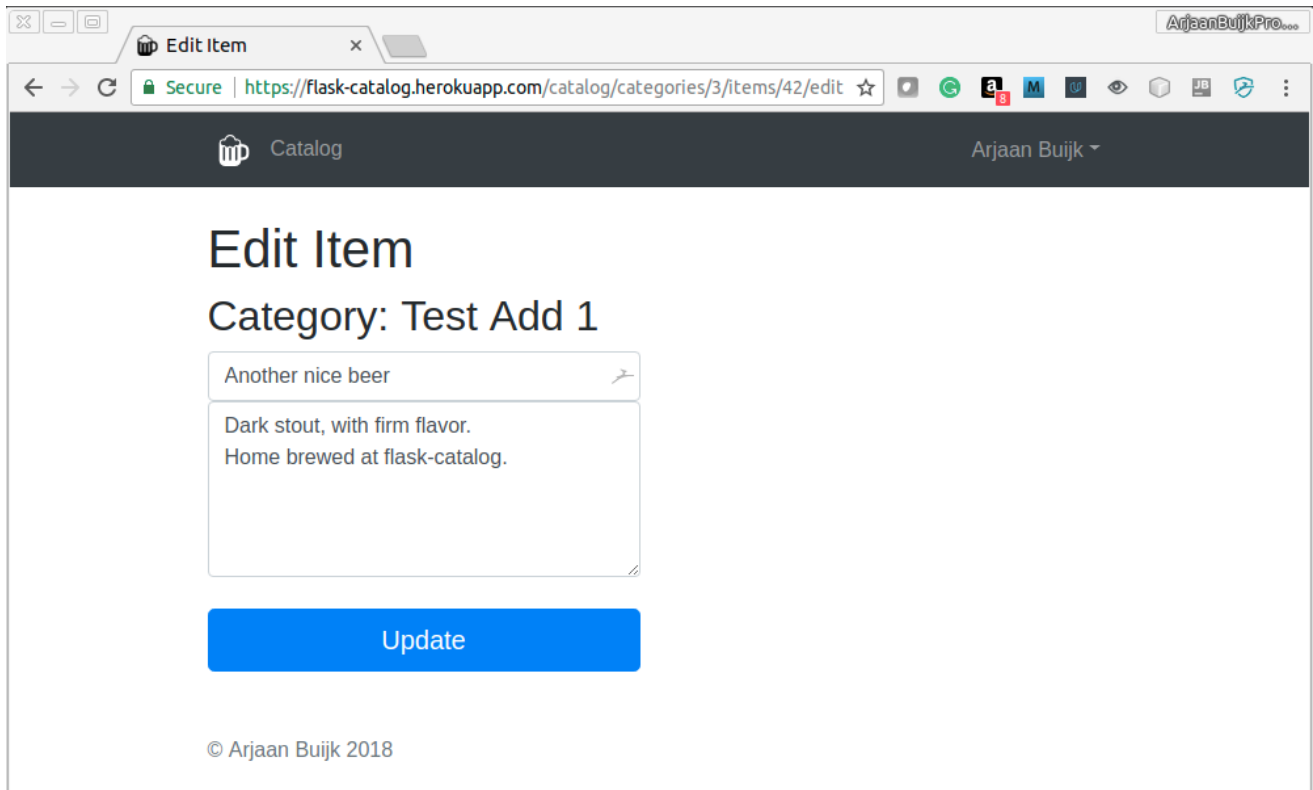
© Arjaan Buijk 2018

Note how the latest beer you add always is automatically selected and shown at the top of the list.



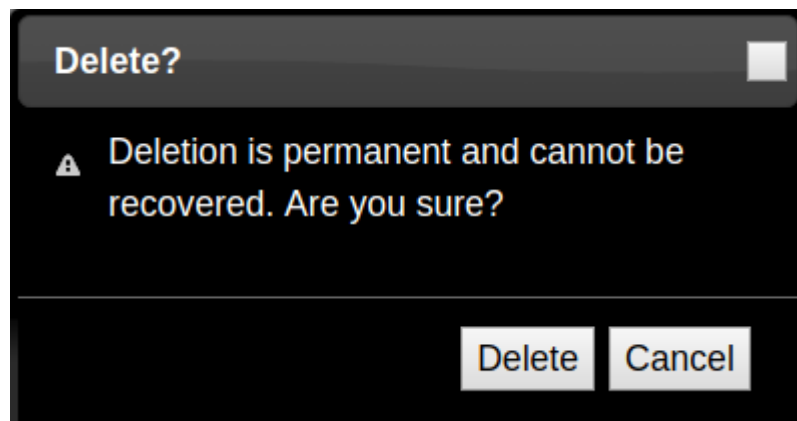
You are allowed to also add new beers to a category that you do not own. This comes with a danger though, because when the owner of the category deletes that category (see below) or decides to delete the account altogether, all the beers of that category will be deleted.

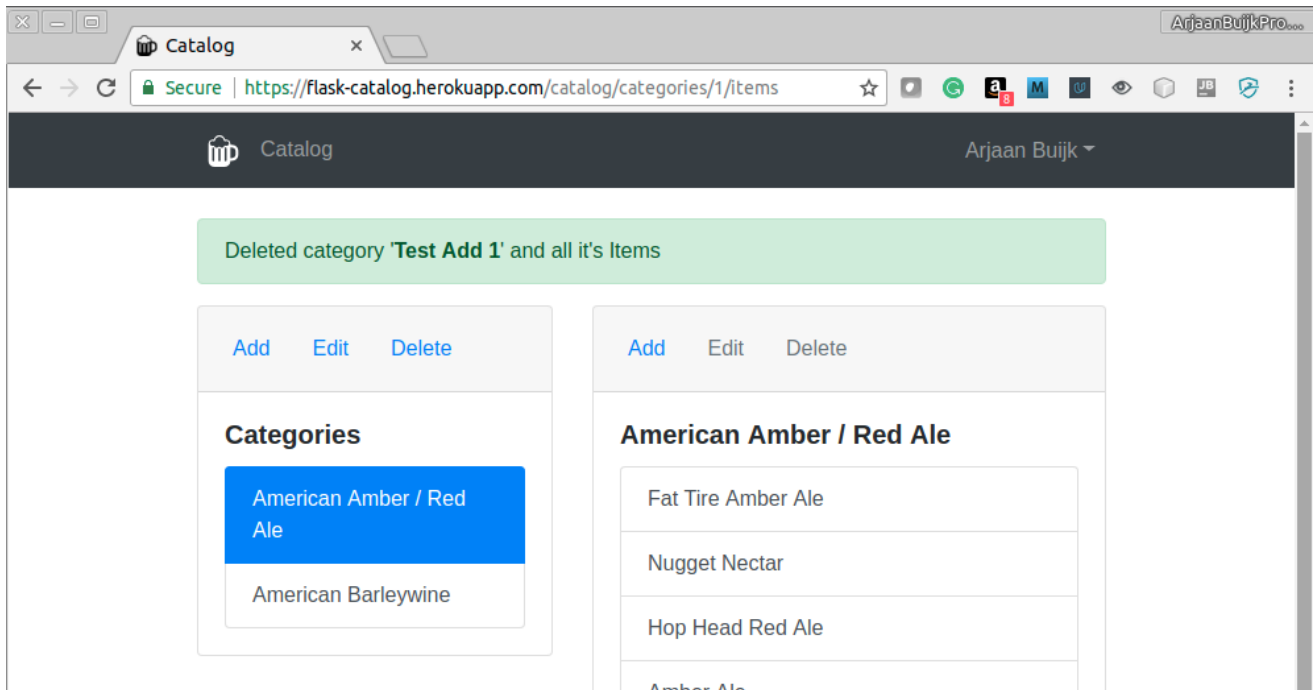
Edit an owned item: <https://flask-catalog.herokuapp.com/catalog/categories/3/items/42/edit>



Any category or item that you own can be edited, as shown here for an item.

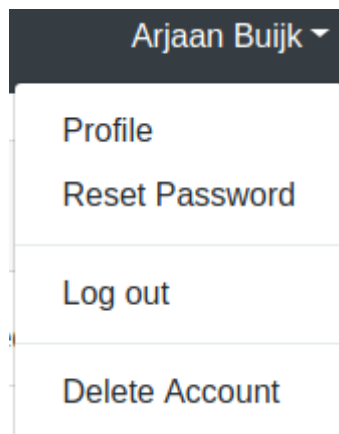
Delete an owned category or item:





When you click on the Delete link for an owned category or item, the application will first ask for confirmation. If you click on Delete, it will go ahead and delete the selected Category or Item. When you delete a category it will also delete all the items in it. Once done, a message is flashed back.

Logged in user dropdown:



As a logged in user, you have access to some user options to update your profile, reset your password, log out or delete your account. Lets try them all out.

Profile: <https://flask-catalog.herokuapp.com/user/profile>


Profile

Secure | <https://flask-catalog.herokuapp.com/user/profile>

Catalog Arjaan Buijk

Profile

Profile Picture



No file chosen

Email

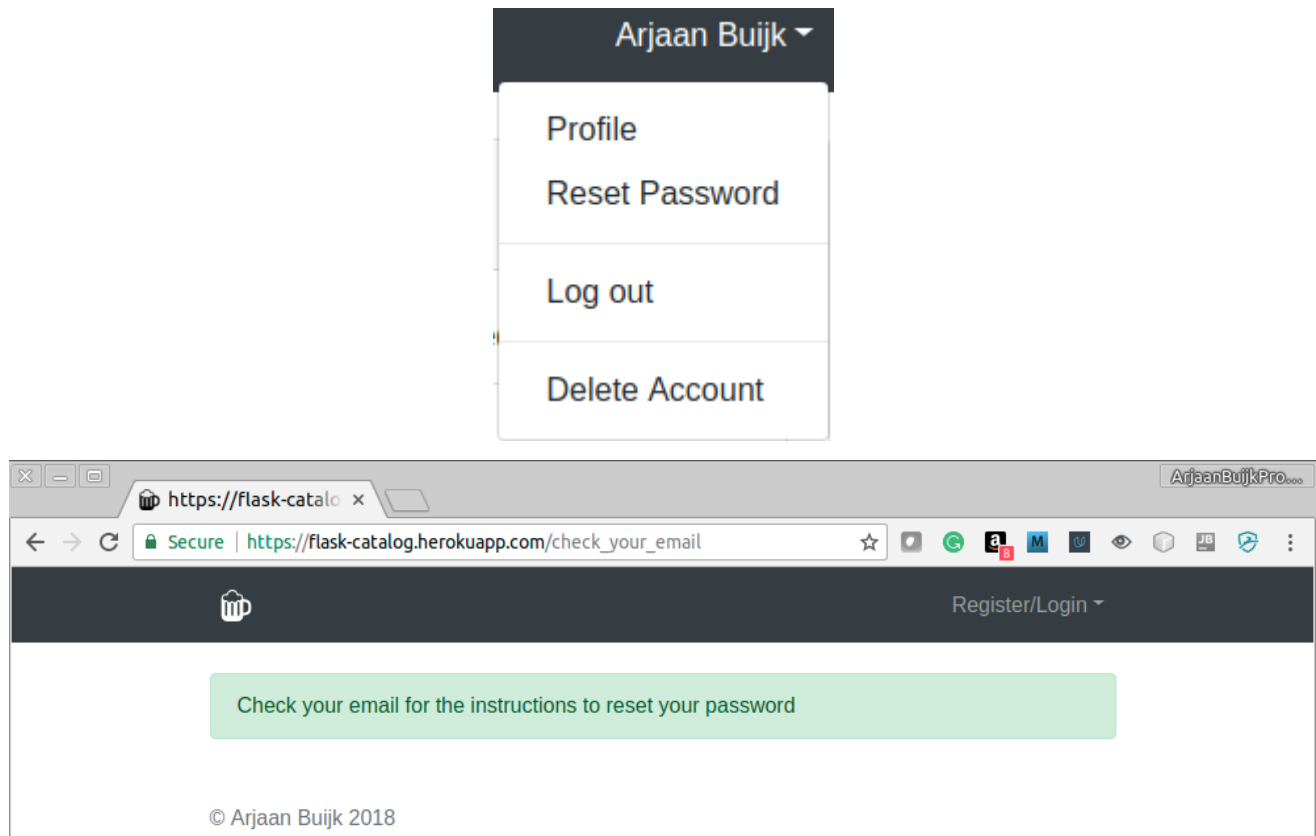
First Name

Last Name

© Arjaan Buijk 2018

On the profile page, the logged in user can update the profile picture, email, first name and last name. Since I logged in with Google, the profile picture that I have stored with my Google account is used, but if I want I can upload another picture.

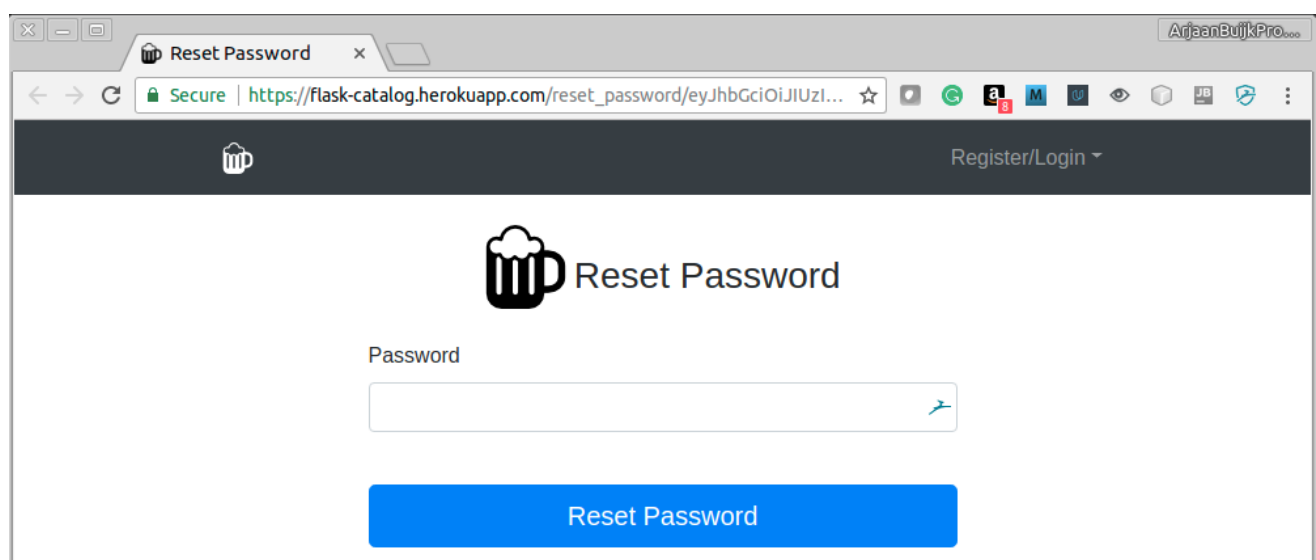
Reset Password:



When selecting Reset Password from the dropdown in the Navbar while logged in, the application will send you an email, and display a message to check your your email, which looks like this:

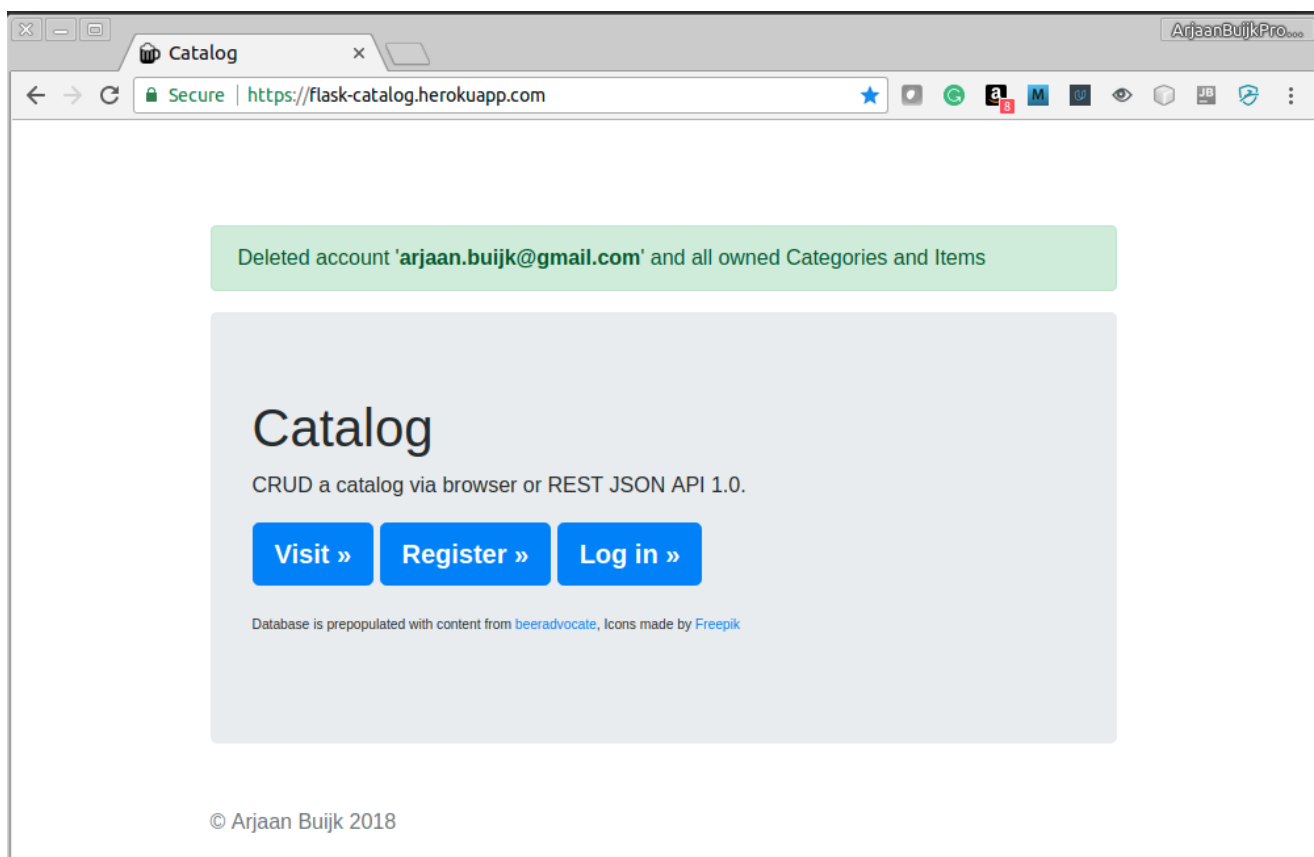
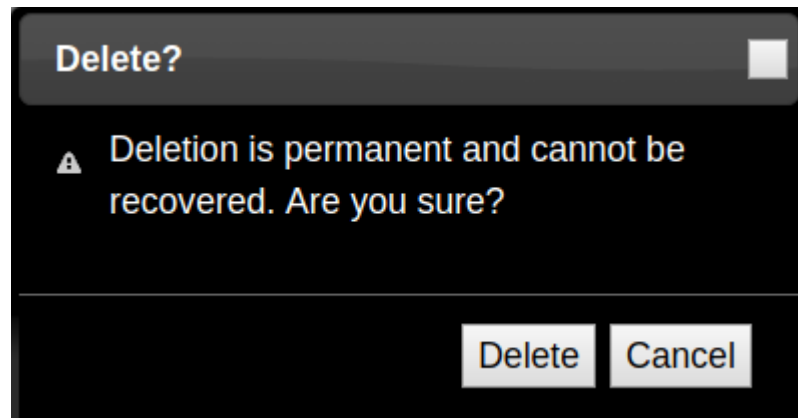
You requested a password reset on Catalog.
Please click the link below to reset your password:

https://flask-catalog.herokuapp.com/reset_password/eyJhbGciOiJIUzI1NiIsImIhdCI6MTUyMzQ4MDM0MywiZmxhZlJoxNTIzNDgzOTQzIiwiaWF0IjeyJyZXNldF9wYXNzd29yZCI6Nn0.xgji6irKes71YfisR2sft-nitVFnzeWwZHZLTi74Fc



When you click on the link in the email, you will be redirected to a form where you can reset your password. Note that in case the registration/login was done with Google OAUTH2, this will be the first time you set your password, so it is not a reset in the literal sense. Once a password has been set, you will then be able to login without Google OAUTH2 and use the applications authentication method.

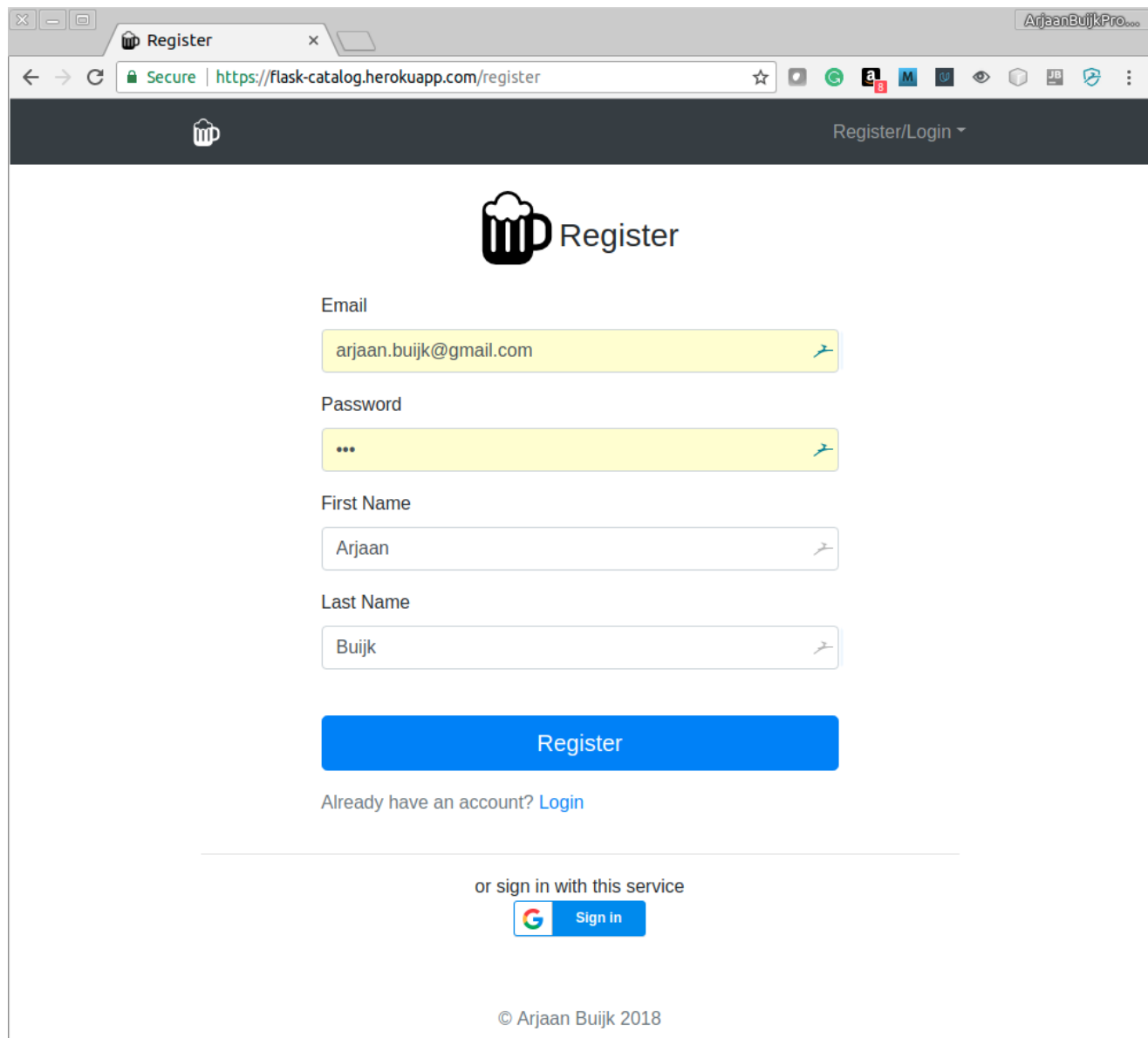
Delete account:



When you select Delete Account, the application will first ask for confirmation. If you click on Delete, it will go ahead and delete the categories you own, including all the items in it (even those you do not own), the items you added to non-owned categories, and finally your user account will be deleted.

Once done, a message is flashed back.

Register: <https://flask-catalog.herokuapp.com/register>



The screenshot shows a web browser window with the URL <https://flask-catalog.herokuapp.com/register>. The page has a dark header with a beer mug icon and a 'Register/Login' dropdown. The main content area is white and features a large beer mug icon followed by the word 'Register'. Below this, there are four input fields: 'Email' (containing 'arjaan.buijk@gmail.com'), 'Password' (containing three dots), 'First Name' (containing 'Arjaan'), and 'Last Name' (containing 'Buijk'). Each field has a small blue icon on the right. Below the fields is a large blue 'Register' button. Underneath the button, it says 'Already have an account? [Login](#)'. At the bottom, there is a section titled 'or sign in with this service' with a Google OAuth2 button labeled 'Sign in'. The footer contains the text '© Arjaan Buijk 2018'.

Email

arjaan.buijk@gmail.com

Password

...

First Name

Arjaan


Last Name

Buijk

Register

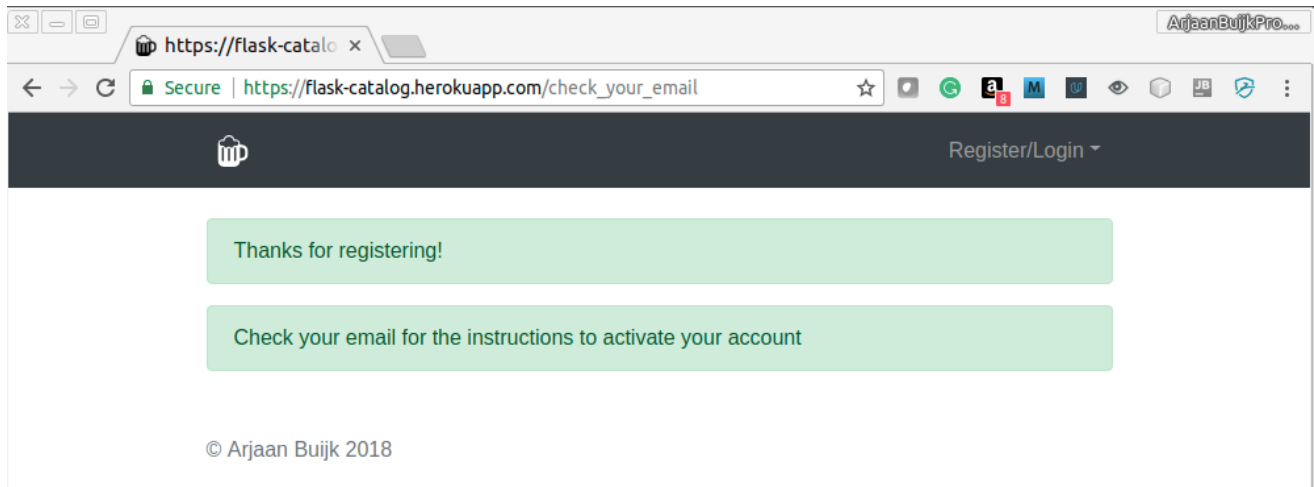
Already have an account? [Login](#)

or sign in with this service

 Sign in

© Arjaan Buijk 2018

After deletion of the account, now lets explore registration without using Google OAUTH2. Click on **Register>>** on the home page, and you will be redirected you to the registration page, where you are asked to provide your email and password together with First Name and Last Name.



When you click Register, your account will be created, but not yet activated. The application will send you an email, and display a message to check your email, which looks like this:

Your account on Catalog was successfully created.
Please click the link below to confirm your email address and
activate your account:

https://flask-catalog.herokuapp.com/confirm/eyJhbGciOiJIUzI1NiIsImIhdCI6MTUyMzU1MjM3MiwiZXhwIjoxNTIzNTU1OTcyfQ.eyJjb25maXJtIjo2fQ.aFVC3eKdfoKD5MdvaleRtfvoYy-5wNe_qad-00EHRbw

Log in


ArjaanBuijkPro...

Secure | https://flask-catalog.herokuapp.com/login?next=%2Fconfirm%2FeyJh...

Log in

Register/Login

Please log in to access this page.

 Log in

Email

arjaan.buijk@gmail.com

Password

...


☐ Remember me

Log in

No account? [Register](#)

Forgot your password? [Reset password](#)

or sign in with this service

 Sign in

© Arjaan Buijk 2018

Catalog

Arjaan Buijk

You have confirmed your account. Thanks!

Add Edit Delete

Categories

American Amber / Red Ale

American Barleywine

Add Edit Delete

American Amber / Red Ale

Fat Tire Amber Ale

Nugget Nectar

Hop Head Red Ale

Amber Ale

When you click on the link in the email, you will first be asked to login, and if that is successful, your account will be activated.

Back End

The API was developed using a very nice flask extension called [Flask-REST-JSONAPI](#). Implementing an API with this extension is a bit more involved than just providing some endpoints that return json, but it gives tremendous capability once finished. For example, it provides automatically:

- JSON responses that adhere strictly to the [JSON API 1.0](#) specification.
- Links in the JSON responses to related object, complying with the REST requirement of *discoverability*.
- Pagination of JSON responses that involve lists.
- Powerful filtering capability.

The implementation of the API are best documented via examples. You can try them out by running this Jupyter notebook from your computer:

```
(venv) $ cd test
(venv) $ jupyter notebook e2e_REST_API_small_demo_heroku.ipynb
```

This will open up the Jupyter notebook in your default browser, and you can run the commands.

You can view an executed notebook, exported as HTML [here](#).

In the examples it is shown that every API call is authenticated, either with user+password, or by means of a JWT token.

To get all the categories you can use this request in python:

```
url = SERVER+'/api/v1/categories/'
headers = get_api_headers(token, '')
r = requests.get(url, headers=headers)
```

Utilities are provided in the Jupyter notebook to print the request and response, and the output is:

```
=====
===THE REQUEST WE SENT===
=====

-----
REQUEST METHOD and URL:
GET https://flask-catalog.herokuapp.com/api/v1/categories/

REQUEST HEADER:
{ 'Accept': 'application/vnd.api+json',
  'Accept-Encoding': 'gzip, deflate',
  'Authorization': 'Basic '
```

```
'ZXlKaGJHY2lPaUpJVXpJMU5pSXNjbWxoZENJNk1UVXlNe1kxTXpJMk5Td2laWGH3SWpveE5USXp0a1UyT0RZMWZ  
RLmV5SnBaQ0k2TTMwLlJaVmg5cUEtTVpFd21IZ1BwaURDMdtU0JvYTBPnUNpUk5Ude92VWQ10Ek6',  
  'Connection': 'keep-alive',  
  'Content-Type': 'application/vnd.api+json',  
  'User-Agent': 'python-requests/2.18.4'}
```

REQUEST BODY:

request.body is empty

```
-----  
=====
```

===THE RESPONSE WE RECEIVED===

```
=====
```

RESPONSE STATUS CODE:

200

RESPONSE HEADER:

```
{ 'Connection': 'keep-alive',  
  'Content-Length': '705',  
  'Content-Type': 'application/json, application/vnd.api+json',  
  'Date': 'Fri, 13 Apr 2018 21:01:23 GMT',  
  'Server': 'unicorn/19.7.1',  
  'Via': '1.1 vegur'}
```

RESPONSE TEXT (Body):

```
{ 'data': [ { 'attributes': { 'name': 'American Amber / Red Ale',  
                             'timestamp': '2018-04-11T21:16:20.990182+00:00'},  
              'id': '1',  
              'links': {'self': '/api/v1/categories/1'},  
              'relationships': { 'user': { 'links': { 'related':  
'/api/v1/categories/1/user',  
                                                    'self':  
'/api/v1/categories/1/relationships/user'}}}},  
              'type': 'category'},  
  { 'attributes': { 'name': 'American Barleywine',  
                   'timestamp': '2018-04-11T21:16:21.099433+00:00'},  
    'id': '2',  
    'links': {'self': '/api/v1/categories/2'},  
    'relationships': { 'user': { 'links': { 'related':  
'/api/v1/categories/2/user',  
                                                    'self':  
'/api/v1/categories/2/relationships/user'}}}},  
    'type': 'category'}],  
  'jsonapi': {'version': '1.0'},  
  'links': {'self': 'https://flask-catalog.herokuapp.com/api/v1/categories/'},  
  'meta': {'count': 2}}
```

Similar examples are provided to get the details of a certain category or item.

pycodestyle

The tool [pycodestyle](#) checks that the code conforms to the [PEP 8 -- Style Guide for Python Code](#).

To verify that all code in the project follows pycodestyle recommendations issue this command:

```
(venv) $ pycodestyle *.py application test
```

No output is provided which means 100% compliance.

Note that I am not checking the code in the migrations folder, because it is automatically generated by the Flask-Migrate extension each time we update the database tables.

Pylint

The tool [Pylint](#) is complementary to pycodestyle, in that it also checks that the code conforms to the [PEP 8 -- Style Guide for Python Code](#), but it finds a lot more issues than pycodestyle. I find it easiest to first comply with pycodestyle, and then fix remaining issues found by Pylint.

To verify that all code in the project adheres to Pylint recommendations issue this command:

```
(venv) $ pylint --rcfile=.pylintrc *.py application test
Using config file <--path-->/FullstackND-Catalog/.pylintrc
-----
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

A perfect score of 10/10 was achieved.

Note that the .pylintrc file was created after installation with this command:

```
(venv) $ pylint --generate-rcfile > .pylintrc
```

And after creation, it was edited to avoid giving some unavoidable errors related to **dynamically generated member attributes of classes and modules**, as described [here](#) and [here](#).

I added:

- session
- scoped_session
- flask_sqlalchemy

```
# List of members which are set dynamically and missed by pylint inference
# system, and so shouldn't trigger E1101 when accessed. Python regular
```

```
# expressions are accepted.
generated-members=session

# List of class names for which member attributes should not be checked (useful
# for classes with dynamically set attributes). This supports the use of
# qualified names.
ignored-classes=optparse.Values,thread._local,_thread._local, scoped_session

# List of module names for which member attributes should not be checked
# (useful for modules/projects where namespaces are manipulated during runtime
# and thus existing member attributes cannot be deduced by static analysis. It
# supports qualified module names, as well as Unix pattern matching.
ignored-modules=flask_sqlalchemy
```

In addition to this Pylint configurations that apply to the whole application, a few specific checks were disabled inside the code, using comment lines like this:

[file: config.py](#)

```
# Accepted pattern in Flask to use a class for configuration
# See: http://flask.pocoo.org/docs/0.12/config/
# pylint: disable=too-few-public-methods
```

In total, I inserted 32 pylint disable statements.

Framework & Extensions

The application is written in [Python 3](#) using the [Flask](#) microframework, with following extensions.

- [Flask-REST-JSONAPI](#)
- [Flask-SQLAlchemy](#)
- [Flask-Bcrypt](#)
- [Flask-HTTPAuth](#)
- [Flask-Login](#)
- [Flask-Mail](#)
- [Flask-Uploads](#)

The front-end HTML pages are generated with:

- [Flask-WTF](#)
- [Bootstrap V4.0.0](#) (*)

The end-2-end test client is written in Python with following extensions:

- [requests](#)
- [Pillow](#)

(*) We do not install Bootstrap V4.0.0, but use a CDN.

Installation

Development and testing was done on Ubuntu 16.04, using Python 3.6.2, to be compatible with environment deployed on heroku.

The steps described here show how to run locally within a python virtual environment.

Step 1. Clone the project repository

```
$ git clone https://github.com/ArjaanBuijk/FullstackND-Catalog.git
```

Step 2. Configure project Define the environment variables in a **.env** file, including the following variables:

```
SECRET_KEY = '<create-a-secret-key>'
BCRYPT_LOG_ROUNDS = <an integer>

# Using gmail (FOR EXAMPLE. REPLACE WITH OWN SMTP SERVER)
MAIL_SERVER = 'smtp.googlemail.com'
MAIL_PORT = 465
MAIL_USE_TLS = False
MAIL_USE_SSL = True
MAIL_USERNAME = '-----@gmail.com'
MAIL_PASSWORD = '#####'
MAIL_DEFAULT_SENDER = '-----@gmail.com'

# We initialize one ADMIN, one USERMANAGER and one USER
ADMIN_EMAIL = '-----'
ADMIN_PW = '-----'
ADMIN_FIRST_NAME = 'Default'
ADMIN_LAST_NAME = 'Admin'

USERMANAGER_EMAIL = 'um@example.com'
USERMANAGER_PW = '-----'
USERMANAGER_FIRST_NAME = 'Default'
USERMANAGER_LAST_NAME = 'Usermanager'

USER_EMAIL = 'us@example.com'
USER_PW = '-----'
USER_FIRST_NAME = 'Default'
USER_LAST_NAME = 'User'

# Uploads

IMAGE_DEST = 'application/static/uploads/img'
DEFAULT_DEST = 'application/static/uploads'

# Google OAUTH2
# You have to register the app with Google yourself and then copy/paste the credentials
# json into this environment variable.
```

```
# Make sure to use TRUE JSON, with " ", not ' '
GOOGLE_OAUTH2_ENV = '{"web":{"client_id":"-----","project_id":"-----",
"auth_uri":"-----","token_uri":"-----","auth_provider_x509_cert_url":"-----",
"client_secret":"-----","redirect_uris":
["http://localhost:5000/categories/"]}}'
GOOGLE_OAUTH2_FILE_PATH = '<just-a-name>.json'
```

Note that for security reasons this configuration file is not included in the github repository.

Step 3. One time: make sure python3-pip and python3-env are installed

```
$ sudo apt install python3-pip
$ sudo apt-get install python3-venv
```

Step 4. One time: prepare the python3 virtual environment

```
$ cd Catalog
$ python3.6 -m venv venv
$ source venv/bin/activate
(venv)
(venv) $ pip install --upgrade pip
(venv) $ pip install -r requirements.txt
```

Alternatively, instead of installing the required python packages using the file 'requirements.txt', which installs the specific versions that were used during development and testing, you can also enter these commands, to install the latest version of each package. Using this script has an advantage in that it also installs and configures the **Jupyter notebook**.

```
$ cd Catalog
$ python3.6 -m venv venv
$ source venv/bin/activate
(venv) $ ./pip_all.sh
```

This installs the following into our virtual environment:

```
-----
#####
## Verify we are running the proper pip ##
#####
which pip
pip install --upgrade pip

#####
## Install flask and all the extensions ##
#####
pip install flask
pip install Flask-REST-JSONAPI
pip install flask-sqlalchemy
pip install flask-migrate
pip install flask-httpauth
pip install flask-login
```



```

pip install flask-mail
pip install flask-uploads
pip install Flask-WTF

#####
## Install additional python packages used by application ##
#####

pip install oauth2client
pip install python-dotenv

#####
## Install python packages for e2e test ##
#####

pip install requests
pip install Pillow

#####
## Install Jupyter notebook for e2e test ##
#####

pip install jupyter
pip install ipykernel
python3 -m ipykernel install --user

#####
## Install PEP8 checkers ##
## - Pylint ##
## - pycodestyle ##
## - pep8 ##
#####

pip install pylint
pip install pep8
pip install pycodestyle

#####
## Install gunicorn webserver ##
#####

pip install gunicorn

#####
## Install package that allows SQLAlchemy to connect to Postgres database ##
#####

pip install psycopg2

```

Step 5. Activate the python virtual environment and start the application server

This will start the application with a clean and fresh database that contains only the default content:

```
$ cd FullstackND-Catalog
$ source venv/bin/activate
(venv) $ export FLASK_APP=catalog.py
(venv) $ ./reset_db_migrations.sh
(venv) $ flask run
```

You will see this output printed to the console:

```
* Serving Flask app "catalog"
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

For end-2-end testing, a Jupyter notebook with the python 3 kernel is used. This must be installed and configured with:

```
(venv) $ pip install --upgrade pip
(venv) $ pip install jupyter
(venv) $ pip install ipykernel
(venv) $ python3 -m ipykernel install --user
```

For validating python syntax and style 3 tools are used, namely **pycodestyle** and **pylint**. These can be installed with:

```
(venv) $ pip install --upgrade pip
(venv) $ pip install pylint
(venv) $ pip install pycodestyle
```

IDE Integration of Pylint and pycodestyle

Since I use Wing IDE 6, I will explain how to setup so called *tool panels*, which makes finding & fixing issues a lot more comfortable.

Integration of Pylint into Wing IDE 6

PyLint is already available as a toolpanel, and setup and activation is described in the [help](#). To avoid some error from cluttering up the result a `.pylintrc` file was created in the root folder of our project. The content of this `pylintrc` file is described above. It is important that you save your wing project file (`.wpr`) in the same folder as the `.pylintrc` file, so that the `.pylintrc` configuration will be automatically used when running PyLint from within the IDE.

Integration of pycodestyle into Wing IDE 6

pycodestyle is not out-of-the box available as a toolpanel, but it is easily installed by using a script written by [Stefan Tjarks](#), as follows:

- Download pep8panel.py from [here](#)
- Store it in the folder ~/.wingide6/scripts
- Modify these lines in file ~/.wingide6/scripts/pep8panel.py

```
.  
PEP8_COMMAND = '<your_path>/FullstackND-Catalog/venv/bin/pycodestyle'  
.  
.  
# See: https://bitbucket.org/stj/pep8panel/issues/3/fix-pep8panel-for-wingide-6  
# from wingutils import location  
from wingbase import location
```

- In Wing IDE 6, select: Edit --> Reload all scripts
- Restart Wing IDE and insert tool panel called pep8