Multi-Purpose Software License Manager Testing

Capstone Project | FEUP | 2022

Isla Cassamo
up201808549

André Malheiro
up201706280

Table of contents

1. Introduction

Handling license models is a necessity for software distributors. Be it the generation of new licenses or the publishing of new products, it is crucial to keep things centralized into one single application. Regardless of the provided limits and various options, licensing your software must be a straightforward task.

Our Multi-Purpose software license manager, developed for BUILT CoLAB, aims to achieve this by creating a web application that handles the distribution of software. To achieve that, we developed a simple interface that offers managing capabilities behind a small learning curve. We accomplish that without sacrificing features and keeping the application efficient.

1.2. Our Added Proposal

In order to guarantee the correct usage and appliance of the functionalities developed in the main application, one must conduct tests to the program to ensure everything is working as expected.

Testing is of most importance as it is a key aspect in the avoidance of security, performance and monetary issues while also performing as an instigator for quality code, satisfaction and development of the project. [1]

Our project seeks to satisfy the crucial need of testing in the application before it reaches the end user upon completion of the project.

2. Development Process

An initial meeting was used to establish the foundations of our work. As the application was built using the Flask microframework [3], the team decided to implement the tests using the python framework *Pytest*[2], written in Python **[4]** and kept our work versioned using Github **[5]**.

The goals of the project were to:

- achieve a greater than 60% coverage of unit tests;
- test the api endpoints;

- automate the tests in the project repository.

The work was divided by the members. Isla was responsible for the unit tests, while André developed the API tests.

Initially, the first step was to set up PyTest in order to run the Application and test it. This was done by creating a configuration file, conftest.py, used by PyTest to configure the application before each test takes place.

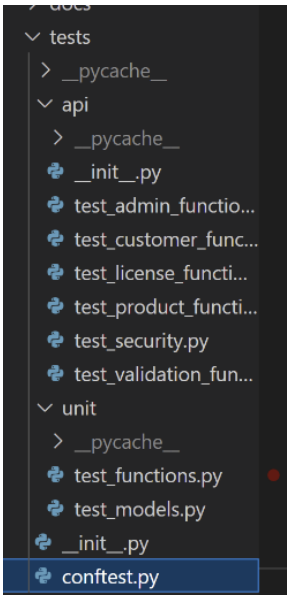
The next step was to prepare the environment for a test and create it. Some tests did not require any preparation, except application setup, and others required specific inputs to the database.

To finish the project, a GitHub workflow was created to automate the tests scripts, as requested by the project proponent.

3. Organization

The tests were done using Pytest and therefore run on the same application folder but not using main. The tests are run on a different call to the system terminal who identifies the test functions by searching for **test_** on the function definitions and running using the **conftest.py** definitions.

We identified two types of significant tests: Unit Tests and API Tests and organized the tree accordingly to best test readability.



3.1. Fixtures

Pytest has an interesting feature named fixtures. These are pieces of code, executed before each test, that create its environment. This means, we use fixtures to define the Application state, the database data - the general environment a test needs before execution. Fixtures also provide us with a method to clean the environment after each test.

In our program the main used fixtures were the definitions of Client, App and Auth. In order to provide testing conditions to our functions these three must be defined as each provides a key point of the app. Other fixtures were also developed, in order to fill the database with the needed data.

Client - Represents an instance of the FlaskClient who performs actions to the app.

App - Instantiates a running version of the application to test on.

Auth – Creates an authenticated instance for the FlaskClient. In the main app, almost every instance of the client needs authentication and therefore it is of most importance.

These three fixtures are defined in **conftest.py,** and, like the remaining fixtures, are 'called' by passing its name as a test function parameter, as seen below.

```
def test_owner_access_admin_list(auth,client):
```

4. Testing

The main goal for testing was the achievement of at least 60% coverage on all fronts. To do so we followed the testing standards **[6]** starting from the best practices and developed a testing unit represented in the following arguments.

4.1. Unit Testing

Unit tests are tests who focus on testing small units of code in isolation. Here Unit Tests were used in order to test the Database Models and also on utility functions that are called viewdly by the program.

The focus for testing was then testing the DatabaseAPI file functions that represented all models and model functionality and all function calls that represented a vital part of the program's functionalities.

Testing was done by creating stubs of the program objects who were the most similar to the real scenarios that will be used by end users and verifying the correct functionality of the functions that affect the object and asserting such changes.

4.1.1. Functionality List

Unit tests were divided in two sets, function testing which are tests to fundamental functions used all over the code and model testing which are tests to the Database accessing functions following the structure below:

- **Model testing (test_model.py)**
    - *test_newuser* - Test creation, find, deletion and other functions associated with the model User.
        - DBAPI.generateUser()
        - DBAPI.obtainUser()
        - DBAPI.toggleUserStatus()
        - DBAPI.deleteUser()
    - *test_changePassword* - Test that asserts the correct change in password by the User. Also tests the correct usages of the hashing functions.
        - DBAPI.changeUserPassword()
        - check_password_hash()
    - *test_new_old_user* - Tests creation, find, edit and deletion of the Customer Model.
        - DBAPI.createCustomer()
        - DBAPI.getCustomer()
        - DBAPI.getCustomerByID()
        - DBAPI.deleteCustomer()
        - DBAPI.modifyCustomer()

    - *test_new_product* - Tests the creation, edit, find, deletion and other functions related to the Product model. This function also tests the creation of product keys.
        - DBAPI.createProduct()
        - DBAPI.getProductCount()
        - DBAPI.editProduct()
        - DBAPI.getProductThroughAPI()
        - keys.generate_product_keys()
    - *test_key_registration* - Tests the creation, edit, find, deletion and other functions related to the Key Model. As both are related functions, it also tests all functions related towards the Registration Model.
        - DBAPI.createKey() - creates a key
        - DBAPI.getKeyBySerialKey()
        - DBAPI.getKeys()
        - DBAPI.addRegistration() - register the key to the product
        - DBAPI.getKeyHWIDs()
        - DBAPI.getDistinctClients()
        - DBAPI.deleteRegistrationOfHWID()
    - *test__admin_logs* - Tests the creation and find of the functions related to the ValidationLog and ChangeLog models.
        - DBAPI.submitLog()
        - DBAPI.getUserLogs()
        - DBAPI.submitValidationLog()
        - DBAPI.queryValidationLogs()
- **Functional Testing (test_function.py)**
    - *test_edits* - Tests the creation and editing the customer and products through high level functions using previously tested database calls to assert.
        - customers.createCustomer()
        - customers.editCustomer()
    - *test_license* - Tests the most important high level function, the creation, editing and deletion of a license (product, key, registration, client)
        - licenses.changeLicenseState()
        - unlinkHardwareDevice()

- *test_utils* - Tests the utility functions of the project, flag testing.
  - utils.validateEmail
  - utils.validadeUsername
  - utils.validatePhone
  - utils.validadeClientID
  - utils.validateMultiple_Customer
  - utils.validatemaxDevices
- *test_keys* -
  - keys.generate_product_keys()
  - generateSerialKey()
- *test_admin* - Tests the high end function of creating and editing an admin user.
  - admins.createAdmin()
  - admins.editAdmin()

The code was not fully tested, leaving about 25% of the functions untested as high-level functions that called on the lower levels were left untouched due to authentication issues.

4.2. API Testing

API Testing focuses on the view functions operations. Here we test nominal conditions, HTTP requests and if the server successfully executes the operations, by verifying the database state in the end.

The tests here were also object oriented where one created an object and asserted the correct functionality of the HTTP request in testing by asserting on the object the changes taken. It is different to Unit testing in the sense where instead of function calls, the tests make API calls to cause upon these changes as the example shows.

```
    auth.login()

    # Tries to list an existent product

    response = client.get("/products/id/"+str(created_product.id))

    assert response.status_code == 200
```

```
    # Tries to list an inexistent product
    response = client.get("/products/id/"+str(created_product.id+1))
    assert response.status_code == 404
```

The API was not fully tested. The 4 endpoints regarding log changes were not tested. The remaining endpoints were tested and verify the work that the API needs to do.

4.2.1. Documentation

The documentation of the API testing functions followed the Numpy docstring documentation. Every function is documented, which can be found in the annexes.

5.Conclusion

Overall, the testing phase returned a 92% coverage which achieved and exceeded the goals set by the team, proving the most vital functionalities of the program are set and the program is ready for usage.

We believe the development of this project was helpful for us to improve our software engineering comprehension. The tests are also useful for any future contributors to the open-source project, as it provides the developer with some tools to test the changes. Essentially it allows for work to be more easily followed upon by others and in turn grow the application more.

We still embraced this project as an essential tool for publishers and developers who wish to manage and contain their own licensing management and develop on it.

6. Future Work

For future work, the goal would be to:

- test the remaining 4 API endpoints;
- improve the depth of the tests developed;
- stress test the application;
- test more use cases.

7. References

[1] Pradeep Parthiban: “Why software Testing” *Indium*, 14 April 2021
https://www.indiumsoftware.com/blog/why-software-testing/
Accessed 5 September 2022.

[2] Pallets: Testing Flask Applications - Pallets, 2010
https://flask.palletsprojects.com/en/2.2.x/testing/
Accessed 6 September 2022

[3] Flask, *Flask — Flask Documentation (2.1.x)*, 1 April 2010
https://flask.palletsprojects.com/en/2.1.x/.
Accessed 12 June 2022.

[4] Python 3.9.13, *Python 3.9.13 - Documentation*, 5 October 2020
https://docs.python.org/3.9/
Accessed 12 June 2022.

[5] GitHub, GitHub: Where the world builds software · GitHub, 2008
https://github.com/
Accessed 2 July 2022.

[6] Patrick Kennedy: Testing Flask Applications with Pytest · TestDriven.io, December 14th 2021
https://testdriven.io/blog/flask-pytest/
Accessed 5 September 2022.

8. Annexes

8.1. Admin Tests Documentation

8.2. Customer Tests Documentation

8.3. License Tests Documentation

8.3.1. test_creation

Description: Tests if API successfully creates a license

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests
app	FlaskApp	The app needed to query the Database
create_product	Product	Product orm object added to the database before the test (fixture)
create_customer	Customer	Client orm object added to the database before the test (fixture)

8.3.2. test_delete_without_associated_device

Description: Tests if API successfully deletes a license when there is no associated devices

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests
app	FlaskApp	The app needed to query the Database
create_license	Key	Key orm object added to the database before the fixture

8.3.3. test_delete_wish_associated_device

Description: Tests if API successfully deletes a license when there are associated devices

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests
app	FlaskApp	The app needed to query the Database
create_license	Key	Key orm object added to the database before the fixture

8.3.4. test_switch_state_no_active_devices

Description: Tests if API successfully switches license state. License has no devices associated

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests
create_product	Product	Product orm object added to the database before the test (fixture)
create_customer	Customer	Client orm object added to the database before the test (fixture)
app	FlaskApp	The app needed to query the Database
create_license	Key	Key orm object added to the database before the fixture

8.3.5. test_switch_state_with_active_devices

Description: Tests if API successfully switches license state when there is 1 associated device

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login

client	FlaskClient	The test client to use for requests
app	FlaskApp	The app needed to query the Database
create_product	Product	Product orm object added to the database before the test (fixture)
create_customer	Customer	Client orm object added to the database before the test (fixture)
create_license	Key	Key orm object added to the database before the fixture
add_device	Registration	Registration orm object added to the database before the fixture

8.3.6. test_invalid_switch_state

Description: Tests if API rejects invalid switches license state requests

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests
app	FlaskApp	The app needed to query the Database
create_product	Product	Product orm object added to the database before the test (fixture)
create_customer	Customer	Client orm object added to the database before the test (fixture)
create_license	Key	Key orm object added to the database before the fixture
add_device	Registration	Registration orm object added to the database before the fixture
licenseID	-	Fixture Parameter
action	-	Fixture Parameter
message	-	Fixture Parameter

8.3.7. test_unlinking_device

Description: Tests if API successfully unlinks device from license

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests

app	FlaskApp	The app needed to query the Database
create_product	Product	Product orm object added to the database before the test (fixture)
create_customer	Customer	Client orm object added to the database before the test (fixture)
create_license	Key	Key orm object added to the database before the fixture
add_device	Registration	Registration orm object added to the database before the fixture

8.3.8. test_invalid_unlinking_device

Description: Tests if API rejects invalid unlink requests

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests
app	FlaskApp	The app needed to query the Database
create_product	Product	Product orm object added to the database before the test (fixture)
create_customer	Customer	Client orm object added to the database before the test (fixture)
create_license	Key	Key orm object added to the database before the fixture
add_device	Registration	Registration orm object added to the database before the fixture
licenseID	-	Fixture Parameter
action	-	Fixture Parameter
message	-	Fixture Parameter

8.4. Product Tests Documentation

8.4.1. test_creation

Description: Tests if API successfully creates a product

Parameters:

Name	Class	Description

auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests
app	FlaskApp	The app needed to query the Database

8.4.2. test_access

Description: Tests if API successfully lists a product info when the product exists

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests
app	FlaskApp	The app needed to query the Database
created_product	Product	Product orm object added to the database before the test (fixture)

8.4.3. test_edit

Description: Tests if API successfully edits a product database entry

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests
app	FlaskApp	The app needed to query the Database
created_product	Product	Product orm object added to the database before the test (fixture)

8.5. Security Tests Documentation

8.5.1. test_unauthenticated_views_access

Description: Tests if API accepts/rejects unauthenticated access to resource

Parameters:

Name	Class	Description
client	FlaskClient	The test client to use for requests

8.5.2. test_unauthenticated_views_access

Description: Tests if API accepts/rejects unauthenticated access to resource

Parameters:

Name	Class	Description
client	FlaskClient	The test client to use for requests

8.6. Validation Tests Documentation

8.6.1. test_device_validation

Description: Tests if API successfully validates a product license

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests
app	FlaskApp	The app needed to query the Database
created_product_1	Product	Product orm object added to the database before the test (fixture)
created_customer	Customer	Customer ORM object added to the database before the test (fixture)
created_license	Key	License ORM object added to the database before the test (fixture)

8.6.2. test_max_devices_fail

Description: Tests if API successfully validates licenses until max devices number reached

Parameters:

Name	Class	Description

auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests
app	FlaskApp	The app needed to query the Database
created_product_1	Product	Product orm object added to the database before the test (fixture)
created_customer	Customer	Customer ORM object added to the database before the test (fixture)
created_license	Key	License ORM object added to the database before the test (fixture)

8.6.3. test_device_validation_on_expired_license

Description: Tests if API successfully rejects validation due to expiration date

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests
app	FlaskApp	The app needed to query the Database
created_product_1	Product	Product orm object added to the database before the test (fixture)
created_customer	Customer	Customer ORM object added to the database before the test (fixture)
created_license	Key	License ORM object added to the database before the test (fixture)

8.6.4. test_device_revalidation_on_valid_license

Description: Tests if API successfully revalidates

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests

app	FlaskApp	The app needed to query the Database
created_product_1	Product	Product orm object added to the database before the test (fixture)
created_customer	Customer	Customer ORM object added to the database before the test (fixture)
created_license	Key	License ORM object added to the database before the test (fixture)
add_device_valid	Registration	Registration ORM object added to the database before the test (fixture)

8.6.5. test_device_revalidation_on_expired_license

Description: Tests if API rejects revalidation

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests
app	FlaskApp	The app needed to query the Database
created_product_1	Product	Product orm object added to the database before the test (fixture)
created_customer	Customer	Customer ORM object added to the database before the test (fixture)
created_license	Key	License ORM object added to the database before the test (fixture)
add_device_valid	Registration	Registration ORM object added to the database before the test (fixture)

8.6.6. test_invalid_validation_input_serial_key

Description: Tests if API rejects invalid inputs for validation

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests

app	FlaskApp	The app needed to query the Database
created_product_1	Product	Product orm object added to the database before the test (fixture)
created_customer	Customer	Customer ORM object added to the database before the test (fixture)

8.6.7. test_invalid_validation_input_api_key

Description: Tests if API rejects invalid inputs for validation – api key

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests
app	FlaskApp	The app needed to query the Database
created_product_1	Product	Product orm object added to the database before the test (fixture)
created_customer	Customer	Customer ORM object added to the database before the test (fixture)
created_license	Key	License ORM object added to the database before the test (fixture)

8.6.8. test_invalid_validation_input_api_key

Description: Tests if API rejects data encrypted using another product public key and if server does not crash

Parameters:

Name	Class	Description
auth	AuthActions	AuthActions class object to use for login
client	FlaskClient	The test client to use for requests
app	FlaskApp	The app needed to query the Database

created_product_1	Product	Product (1) ORM object added to the database before the test (fixture)
created_product_2	Product	Product (2) ORM object added to the database before the test (fixture)
created_customer	Customer	Customer ORM object added to the database before the test (fixture)
created_license	Key	License ORM object added to the database before the test (fixture)