**Introduction:**

This application is built with Python and Flask, and it allows an admin user to login using their credentials and manage the developer's assets and licenses. The application uses a MySQL database to store and retrieve data.

**Important:**

GitHub repository: <https://github.com/ecaracasdev/python-mysql.git>

Additionally, the GitHub repository includes a README file that provides instructions for installing the project. It is recommended to refer to the README file to ensure a smooth installation of the application.

Public Url: <http://3.137.187.127:5000/login>

Postman:

In the root folder of the repository you will find a file called :

**python-flask-mysql.postman\_collection.json**

After importing the postman collection you will find all the necessary endpoint with the body and headers detailed. In the usage section of this document you will also find the step use these endpoints

**Entity Attributes:**

**Asset Attributes:**

id: string | number

brand: string

model: string

type: enum<string> ['laptop', 'keyboard', 'mouse', 'headset', 'monitor']

**License Attributes:**

id: string | number

software: string

**Developer Attributes:**

id: string | number

fullname: string

active: Boolean

**DATABASE:**

For this application, I used a different approach to showcase my ORM skills. I chose to use MySQL as a relational database and designed a schema with four main tables: "user" (which contains only the admin user's information, whose credentials I will share via email), a table for "developers", "licenses", and "assets". In this case, I used a many-to-many relationship since a developer can have one or more assets, and an asset can belong to one or more developers (considering that the ACME company may have many assets of the same brand and model). Similarly, for licenses, a developer can have one or many licenses, and a license can belong to one or many developers.

To make data management easier when starting the app, I included an SQL schema that inserts sample values into the tables and creates the appropriate relationships. In case the insertion does not execute correctly when running the app, I will also include steps to insert them manually.

**API Endpoints:**

**Assets:**

GET {base\_url}/assets/add : Returns a list of all assets.

DELETE {base\_url}/assets/{assetId}/delete : Deletes an asset with the given asset ID.

POST {base\_url}/assets/ : Creates a new asset.

**Licenses:**

GET {base\_url}/licenses/: Returns a list of all licenses.

DELETE {base\_url}/licenses/{licenseId}/delete : Deletes a license with the given license ID.

POST {base\_url}/licenses/add : Creates a new license.

**Developers:**  
  
POST {base\_url}/developers/add : This endpoint adds a new developer

GET {base\_url}/developers/ : This endpoint retrieves a list of all the active developers

GET {base\_url}/developers/{dev\_id} :

This endpoint retrieves a single developer's details by ID. You can access this endpoint by making a GET request to {base\_url}/developers/{dev\_id}, where {dev\_id} is the unique identifier of the developer you want to retrieve.

Optionally, you can include a query parameter called property in the request with a value of asset or license. If the property query parameter is set to asset, the endpoint will return a list of all assets that belong to the developer. If property is set to license, the endpoint will return a list of all licenses that belong to the developer.

To summarize:

To get a developer's complete details, make a GET request to {base\_url}/developers/{dev\_id} without any query parameters.

To get a developer's assets, make a GET request to {base\_url}/developers/{dev\_id}?property=asset.

To get a developer's licenses, make a GET request to {base\_url}/developers/{dev\_id}?property=license.

POST {base\_url}/developers/deactivate/{dev\_id} : This endpoint deactivates (or soft-deletes) a developer from the system

POST {base\_url}/developers/addassets/{dev\_id} : This endpoint adds new assets to a developer's list of assets

POST {base\_url}/developers/addlicenses/{dev\_id}: This endpoint adds new licenses to a developer's list of licenses

POST {base\_url}/developers/removeassets/{dev\_id} : This endpoint removes assets from a developer's list of assets.

POST {base\_url}/developers/removelicenses/{dev\_id} : This endpoint removes licenses from a developer's list of licenses.

**USAGE:**

After successfully running the project following the detailed instructions in the README.md file of the repository, these are the steps to follow for using the app.   
  
In the case of this project, I had some issues when running the database migration script, the tables are created correctly but the values are missing. If that is the case when you run the project, please insert the following test values into the DB using the following command

INSERT INTO developer (id, fullname, active)

VALUES (1, 'John Doe', true),

       (2, 'Jane Smith', false);

INSERT INTO license (id, software)

VALUES (1, 'Visual Studio Code'),

       (2, 'PyCharm');

INSERT INTO asset (id, brand, model, type)

VALUES (1, 'HP', 'Elitebook', 'laptop'),

       (2, 'Logitech', 'K120', 'keyboard');

INSERT INTO developer\_licenses (developer\_id, license\_id)

VALUES (1, 1),

       (2, 2);

INSERT INTO developer\_assets (developer\_id, asset\_id)

VALUES (1, 1),

       (2, 2);

**Note: I will provide you a postman collection where you can find example of how to follow this steps with the proper entries and responses**  
  
1. Log in using the admin credentials: username = admin, password = admin.

2. Check the list of assets and licenses using the getAssets or getLicenses endpoint.

3. Check the list of developers using the getDevelopers endpoint.

4. Add more developers, assets, or licenses if necessary.

5. Retrieve information about a developer using their ID:

5.1 Use the same endpoint but pass queryParams as "assets" or "licenses" to retrieve a list of assets or licenses associated with the developer.

6. Use the endpoint to deactivate a developer by their ID to ensure that all their assets and licenses are removed.

7. Use the endpoint to update the list of assets or licenses associated with a given developer.

8. Verify that the assets or licenses were added or removed successfully by checking the getDevByID endpoint.