**Household Hub project detailed description**

# Quick Overview

This project aims to develop a system for managing household devices. The system will allow users to:

* Register device information (name, category, make, model, purchase date, etc.)
* Store documents related to the device (guaranty documents, receipts, etc.)
* Add notes and other relevant details to the device.
* Quickly find a device.
* Manage device categories.

This project provides a centralized platform for tracking and managing household devices, making it easier for users to organize their device information.

This project is a web application developed with Python and Flask.

# Definitions

**Flask:** Flask is a lightweight web application framework for developing web applications in Python. Flask is small and easy to learn.

**Device:** It could be any device. It could be a TV, a car, a gaming console, a mug, etc. It could be anything that the user may want to keep track of. A device may belong to a category.

**Category:** It is a classification for a device. Categories are fully customized. Some categories include Computers, Phones, Gaming Devices, etc. The user can add/edit/remove categories.

# Installing and running the project

1. To run the project, make sure that Python is installed on the computer. The installer can be downloaded from here: [Download Python | Python.org](https://www.python.org/downloads/).
2. Currently this project only depends on a Python package named [Flask](https://flask.palletsprojects.com/en/stable/). To install Flask, you can run the command **pip install flask.**
3. To run the project, open a terminal. Point to the HouseholdHub directory and run the command **python app.py** (see image 1).That command will initialize the Flask web application. In addition, it will list several URLs that can be used to launch the application locally. Pressing Ctrl and clicking the URL will open the browser and launch the website.

A screen shot of a computer code

Description automatically generated

Image 1 Running the project.

# User interface

Launching the website will land on the home page of the Household Hub application. The home page will display a small dashboard with numbers related to the registered devices. For example, in Image 2 below you can see the home page rendering two sections. One section counts the number of devices per category. In addition, another section will tell how many devices are there according to their warranty expiration status. Some devices may have their warranty expired, others may expire within a year, etc. The home page and all other pages will always contain three menus: Home, Find Devices, and Manage Categories.

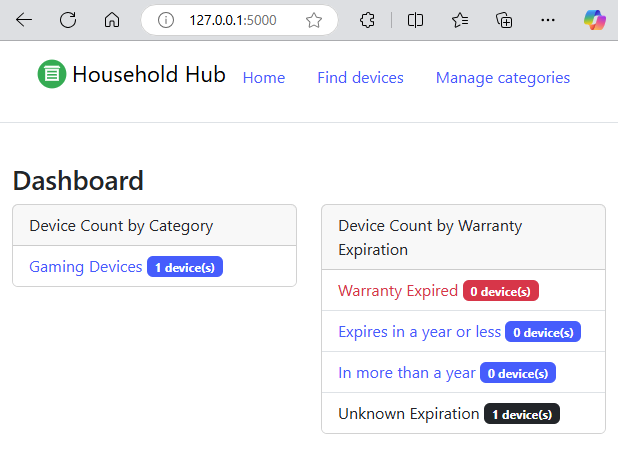


Image 2 Home Page

Clicking on ‘Manage Categories’ will render a page where the user can see all registered device categories. In addition, the user will be able to add categories, delete categories, or edit a category (see image 3). Clicking on the ‘Add Category’ button will render a page where the user can add a new category. At this moment, the only information supported regarding a category is the name (see image 4). Clicking on the edit button next to a category will allow the user to edit the category name. Clicking on the delete button next to a category will delete the category only if the category is not in use by a device.

A screenshot of a device

Description automatically generated

Image Manage Categories

A screenshot of a computer

Description automatically generated

Image Add/Edit Category

Clicking on “Find devices” will render a page where all the devices are listed (see image 5). The Devices page has two sections. The top section is a set of filters where the user can filter devices by name and/or category and/or Warranty expiration status. Setting one of the filters will refresh the device listing according to the filter set. Under the filters section, you will find the device listing section where all the registered devices will be listed. Each device has the option to edit its information or remove the device. This page also has a button to add a brand-new device.

A screenshot of a computer

Description automatically generated

Image Find Devices

Adding a new device will render the page in image six. From there, the user can input the device information such as name, category, purchase price, warranty notes, warranty expiration date, and related documents and links.

A screenshot of a computer

Description automatically generated

Image Adding a new device.

After a new device is added, the application will navigate back to render all devices. Displaying the new device added and a confirmation message indicating that a new device was added (see Image 7). Editing an existing device has a similar flow.

A screenshot of a device

Description automatically generated

Image Rendered devices after a new device was added.

# High-level project technology

This project was developed with Python and Flask. Since this project is a web application, other languages are also incorporated. The user of this web application will receive pages rendered in HMTL as well as some CSS and JavaScript. On the server side, [Flask](https://jinja.palletsprojects.com/en/stable/) is used to handle the requests and responses. Most of the requests are handled, validated, processed, and rendered in HTML files which contain a mix of HMTL, and another language called Jinja. Jinja contains a template syntax that allows for building dynamic pages. For example, with Jinja, a list can be iterated and a ‘dynamic’ table in HTML can be represented.

# Folder structure

* **Root folder:** Contains all other folders that encapsulate different modules for this project. The root folder also contains the file app.py which includes the project initialization.
* **DB folder:** Contains the “database” for this project. For now, the database consists of two JSON files. One file will be “Categories.json” which is a file that contains the definition of all the categories. Another file is “Devices.json” which includes the definition of all the devices. Inside the db folder, there is another folder named docs, which contains all the documents each device may reference.
* **Static folder:** As its name implies, this folder contains static resources. Flask will automatically search this folder to serve static files when needed. Examples of static resources could be images, icons, CSS, etc.
* **Templates folder:** It contains the HTML content that will be rendered in the client’s browser. When a certain URL is requested (for example http://127.0.0.1:5000/my\_categories), Flask will match that route to a function defined, and that function will perform certain logic and request to render a template inside the templates folder.
* **Routes\_bluprints folder:** It contains different modules that provide a logical organization of the site request and response logic. Blueprints help to group related functionality by module. Each blueprint module defines an object of type BluePrint which is then registered as a blueprint in app.py.
* **Business\_rules folder:** It contains two types of modules: Managers and Entities. Manager files contain a class that is responsible for interfacing with the database (a JSON file). Managers will read/write from/to the database and fill up Entity classes (Entity modules/Classes). Blueprint modules will always interface with the business rules’ manager classes. The business\_rules folder provides components that are an abstraction layer from the database.

# Sample request (Every component working together)

One sample workflow comes as follows: When the user clicks on the menu named “Manage categories;” He/She will click on a hyperlink with the URL http://127.0.0.1:5000/my\_categories. That request (also known as route) is registered in the categories\_blueprint.py (see image 8). The categories blueprint is registered on the file app.py which is the Flask web application initialization file (see image 9). The route “/my\_categories” is matched to a function named “my\_categories” via a decorator. This means that every time the request to the route “/my\_categories” is made, Flask will execute the function my\_categories located in the categories\_blueprint.py file. The Python function my\_categories is wrapped inside a try-except block which allows to capture any error that may happen. If an error happens, an error page will be rendered, and the application will not crash (see handle\_route\_error for more information). Inside the try-except block, a static method named get\_all\_categories from the class Categories\_Manager will be called. The get\_all\_categories method will return a list of all categories from the “database” and the variable named categories\_col will receive that returned list. Finally, if everything went right, the function my\_categories will return. The return value will be the call to another Flask function named render\_template. That function receives two parameters: the name of the template that will be rendered (in this case ‘my\_categories.html’) and the categories\_col variable. At the end of this process, the client’s browser will render a page like the one in Image 3.

A screen shot of a computer program

Description automatically generated

Image /my\_categories route in categories\_blueprint.py module

A screen shot of a computer program

Description automatically generated

Image App.py