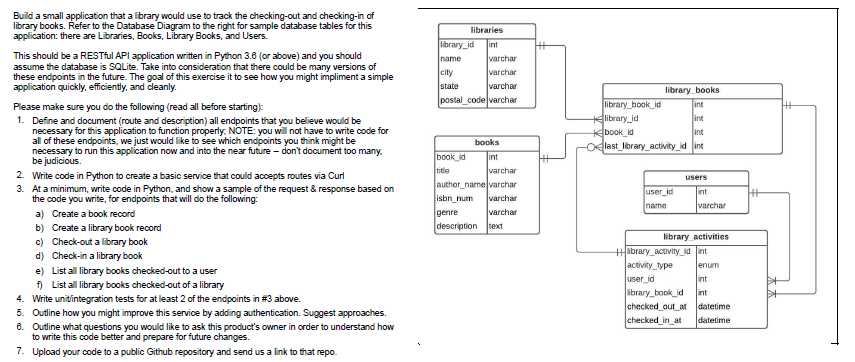
# Library Management Application Release Notes (Initial Version)

## Full Requirements Specifications



# Scope of Work and Details on Tasks Resolution

# 1. Endpoints Definitions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N°** | **Endpoint** | **Route** | **HTTP Methods** | **Description** |
| 1 | books\_blueprint.route: '/add' | books.add | methods=['GET', 'POST'] | create a new book record (books table) |
| 2 | books\_blueprint.route: '/list' | books.list | methods=['POST'] | List existing books from table books |
| 3 | books\_blueprint.route: '/delete' | books.delete | methods=['GET', 'POST'] | Delete a book from books table |
| 4 | librarybooks\_blueprint: '/add' | library\_books.add | methods=['GET', 'POST'] | Create a new 'library book' record (library\_books) |
| 5 | users\_blueprint.route: '/add' | users.add | methods=['GET', 'POST'] | Create a new user (users table) |
| 6 | users\_blueprint.route: '/list' | users.list | methods=['POST'] | List users |
| 7 | users\_blueprint.route: '/list\_by\_act' | users.list\_by\_act | methods=['GET', 'POST'] | List Checkedout Books for a specific user |
| 8 | users\_blueprint.route: '/list\_by\_checkout' | users.list\_lib\_checkout | methods=['GET', 'POST'] | List checkedout Books of a specified library |
| 9 | users\_blueprint.route: '/delete' | users.delete | methods=['GET', 'POST'] | Delete a user (users table) |
| 10 | libraries\_blueprint.route: '/add' | libraries.add | methods=['GET', 'POST'] | Add a library (libraries table) |
| 11 | libraries\_blueprint.route: '/list\_libraries' | libraries.list\_libraries | methods=['POST'] | List Libraries |
| 12 | libraries\_blueprint.route: '/delete' | libraries.delete | methods=['GET', 'POST'] | Delete a Library |
| 13 | libraryactivities\_blueprint.route: '/add' | library\_activities.add | methods=['GET', 'POST'] | Create a New library\_activities record |

# 2. Basic Python Service

Please refer to the Python-Flask based Library Management Application on its base version at:

Public GitHub Repository: <https://github.com/andiebalverde/librarymanagement.git>

## Project Launch Recommendations:

- For practicity, the sql file and migration folder are uploaded to main branch, in case a demo takes place using existing data.

- If the user’d rather start from scratch, it is recommended to remove the sql file and migration folder and run the commands:

flask db init | flask db migrate | flask db upgrade

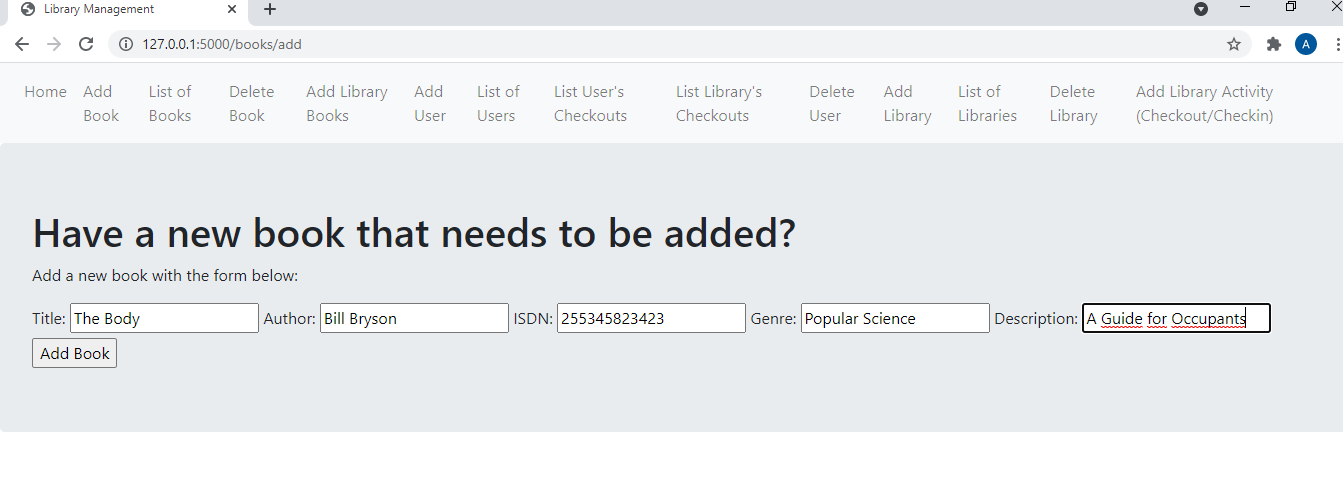
- Set Environmental Variable FLASK\_APP=app.py

- Run using command: flask run

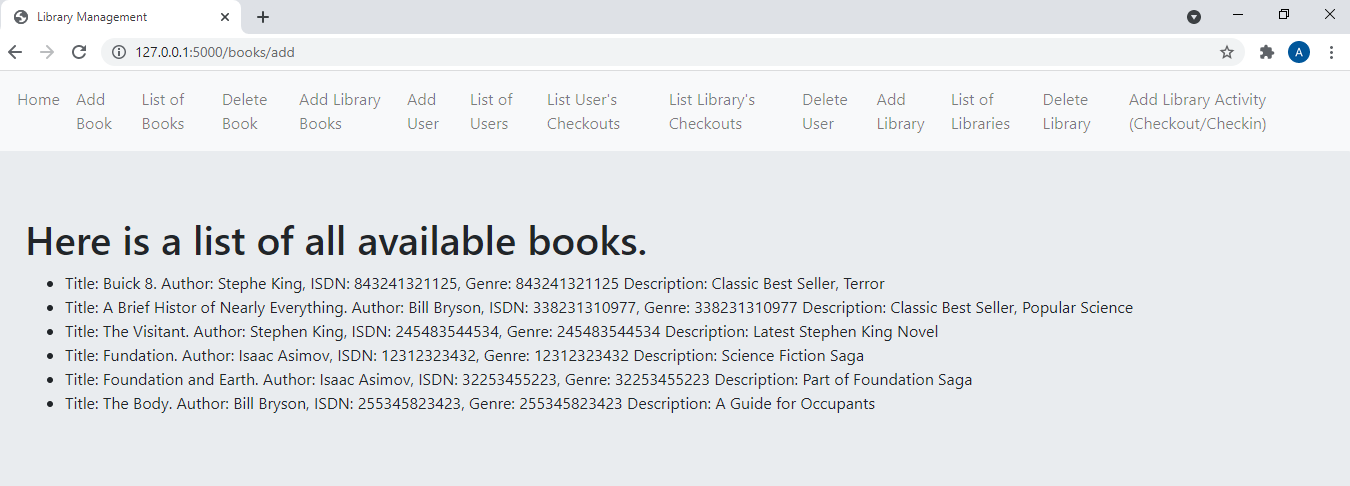
3. Main Endpoints: Sample Requests & Respones

1. Create a Book Record:

Request: Add Book.

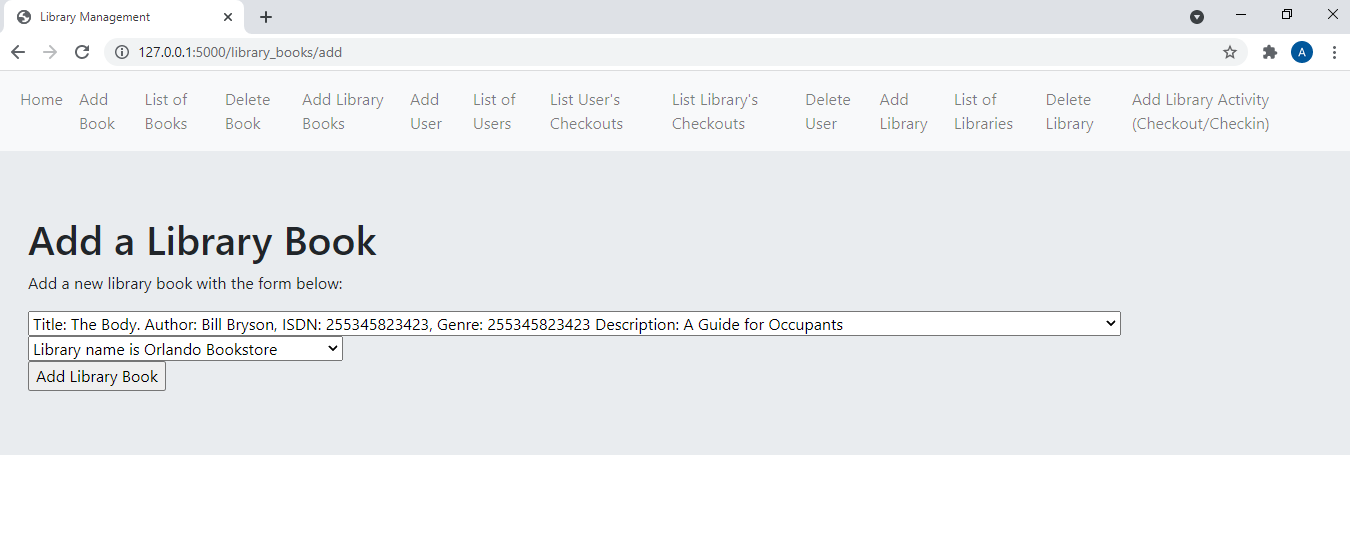


Response: lists all available books including the newly added one.

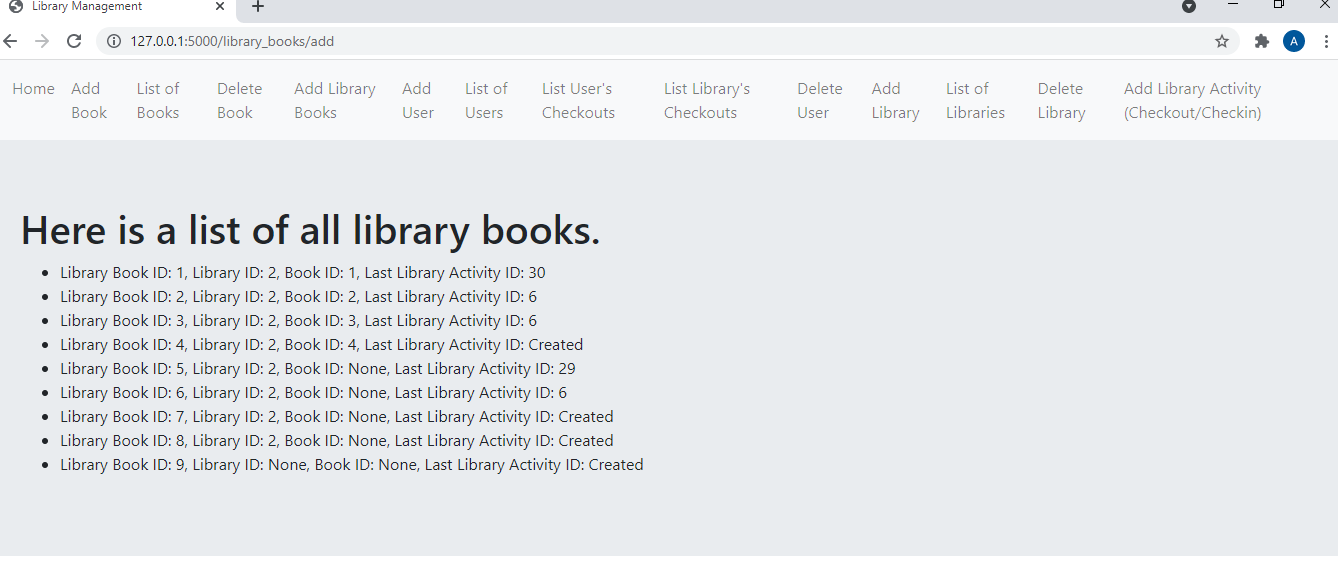


1. Create a Library Book Record

Request: Add Library Books

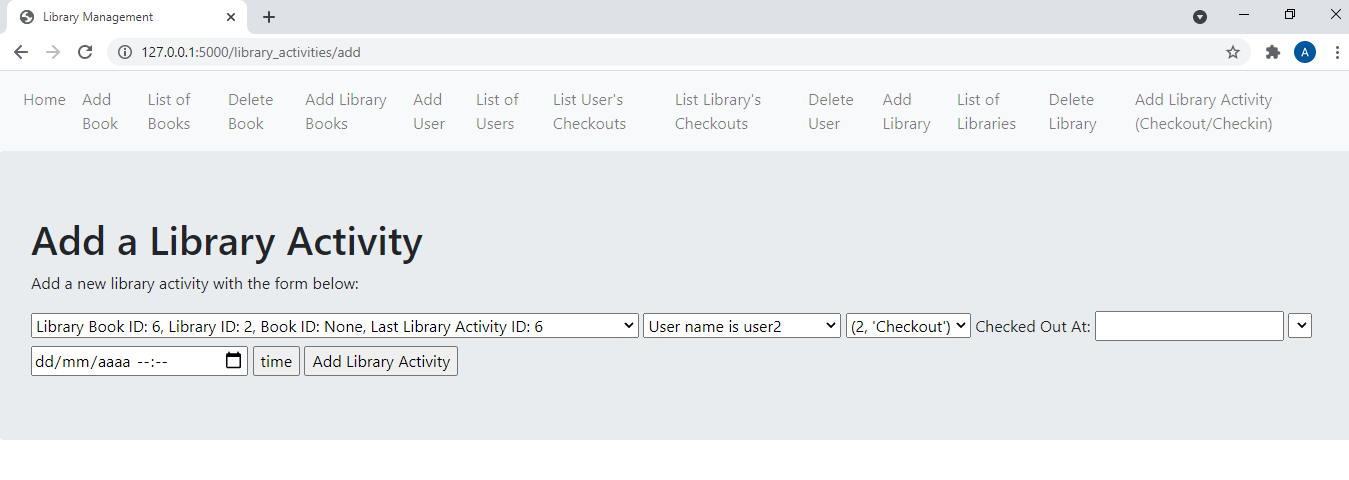


Response: lists all library book records including the newly created one.

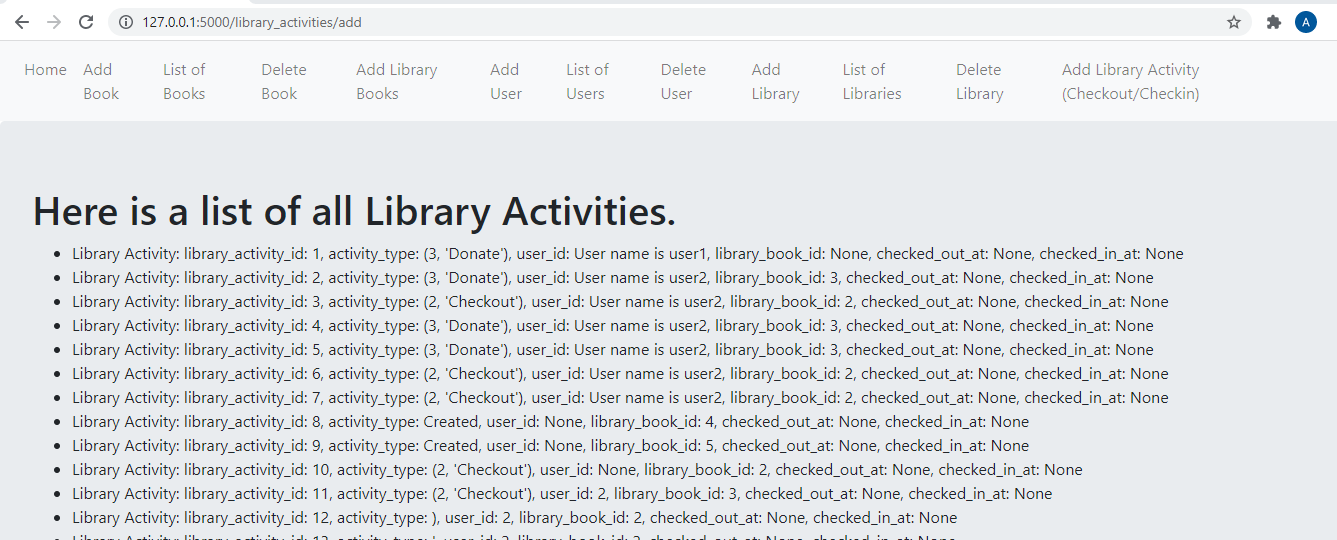


# 3. Checkout a Library Book

Request: Add Library Activity (Checkout/Checkin): select Library Activity “Checkout” for a specific Library Book ID and for a specific user (check out at –datetime- is optional).

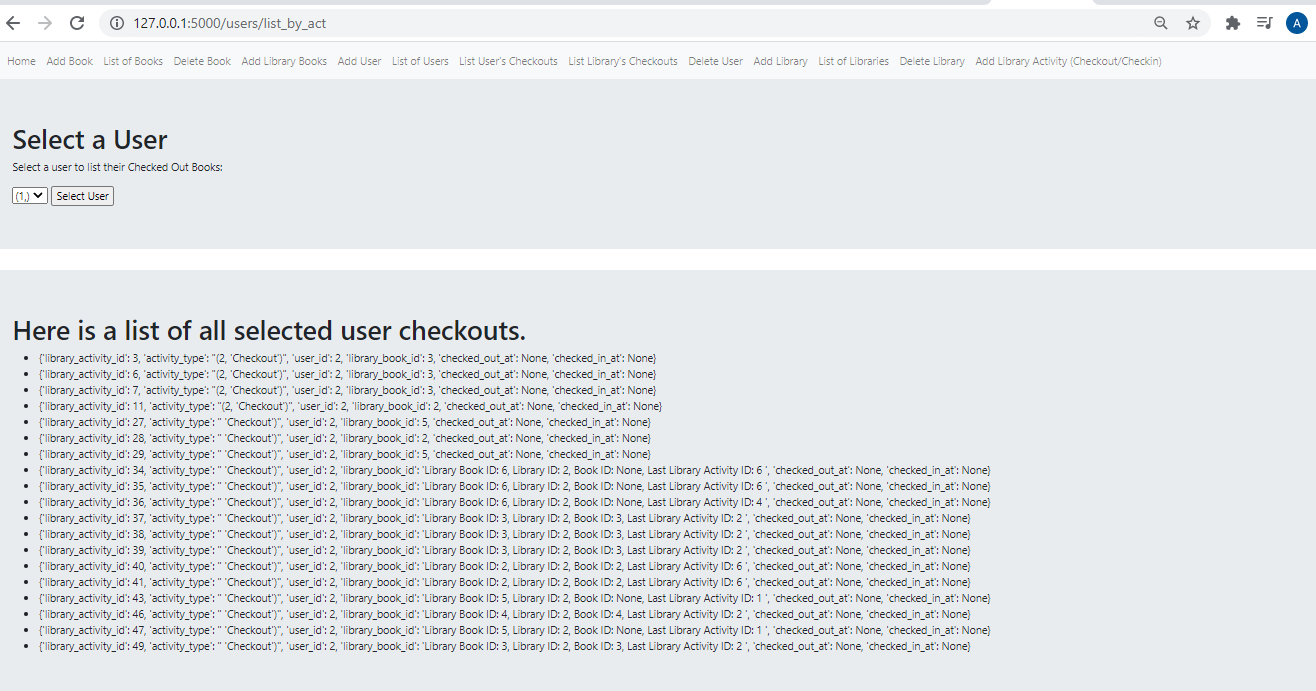


Response shows a list of all library activities, the latest being the one that was just commited:



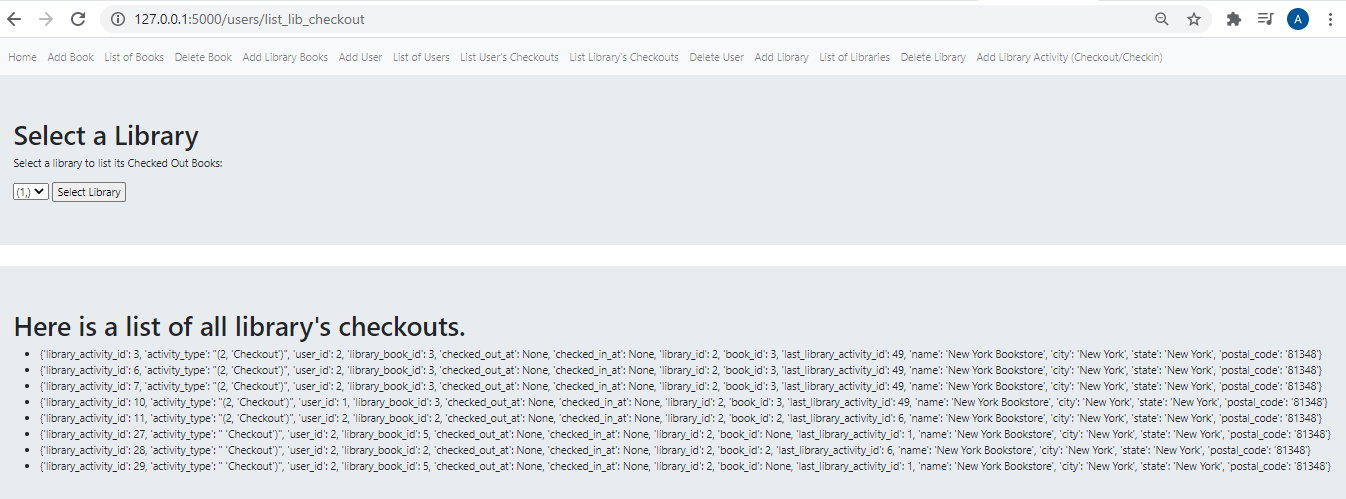
1. Check in library book: same request/response as above, except for the selection of Checkin in the activity field.
2. List User’s Checkouts:

Request: in List User’s Checkouts, specify a user to list their checkouts. As a response a list of all user activity is obtained.



1. List All Library Books Checkedout to a Library

Request: in List Library’s Checkouts, select a library. As a response, a list of all checked out books for that library should be listed.



# 4. Write Unit Tests for Some of the Endpoints

Unit Tests are designed using pytest library. Requirements involve: requests, pytests and eventually jsonschema for more complex test cases, all of these requirements are part of requirements.txt

Under myproject/unit\_tests.py a set of unit tests are designed to verify the healthy of main endpoints:

1. test\_list\_users\_check\_status: verifies the health of “List of Users” endpoint (users.list)
2. test\_list\_user\_checkouts: verifies the health of “List User’s Checkouts” endpoint (users.list\_by\_act)
3. test\_list\_library\_checkouts: verifies the health of “List Library’s Checkouts” endpoint (users.list\_lib\_checkouts)
4. test\_add\_library\_checkouts: verifies the health of “Add Library” endpoint (libraries.add).
5. test\_add\_library\_activity: verifies the health of “Add Library Activity (Checkout/Checkin)” endpoint (library\_activities.add).

Some examples on running unit tests and evaluating results:

From folder \myproject>pytest unit\_test.py::test\_list\_users\_check\_status



From folder \myproject>pytest unit\_test.py::test\_list\_user\_checkouts

# 5. Authentication Outlines for the Service

Simplest Login options involving integration with services like Google (Gmail) or Facebook, are a very good choice for simple applications with a moderate data sensitivity level, and facilitatating sign in / sign on mechanisms for their users, making the service user friendly and secure at the same time.

Since we are working with Flask, the library I recommend is flask-dance

<https://flask-dance.readthedocs.io/en/latest/quickstarts/google.html>

Considering Google Integration login, these are the steps to follow starting from scratch:

1. From Google Developers Console [https://console.developers.google.com](https://console.developers.google.com/)
2. Create a new project.
3. From “APIs & auth” section, click “Credentials”
4. Click the “Create a new Client ID”.
5. Select “Web Application” (application type)
6. Click the “Configure consent screen” button.
7. Fill in application information, click Save.
8. Once you’ve done that, you get: “Authorized JavaScript origins” and “Authorized redirect URIs”.
9. Put <http://localhost:5000/login/google/authorized> into “Authorized redirect URIs”, and click “Create Client ID”.
10. Save the “Client ID” and “Client Secret” since those will be used for the application.

A usage simple example is:



This would involve the development of the following endpoints/templates:

@app.route('/welcome'): will verify if the user is logged in Google, will return an error elsewise.

@app.route("/login/google"): will redirect to google login to validate the application for single sign on for the user.

## 6. Questions for Product Owner

1. Is our client a private or public book library ‘loan’ service?
2. Related to 1, will there be a charge for every book checkout? Questions 1 and 2 may imply to design payment methods, i.e: Stripe Payments (pip install stripe)
3. Is there a time limit for a book checked out? This is related to the pricing schema: there should be a definition of the pricing: is it a flat value per day / per checkout? Is there a fine when the checkin deadline is not accomplished?
4. Will the users (clients checking out/in books) login in the future? Current version is designed in such way that the “users” concept implies the “book users” or “clients”.
5. Currently there’s no “application users” or different user types and privileges (Administrator users for creating new books or library\_books, etc., or Client Users with the ability of performing checkout/checkin activities for specific libraries). Will this be the user management evolution of the service?
6. There could be several units of exactly the same book (same ISDN Number) for same libraries or assigned to different libraries. Should we evolve the database architecture to involve this Quantity concept for every book ID?

## 7. Code Uploaded to Github Repository

Public GitHub Repository: <https://github.com/andiebalverde/librarymanagement.git>

Please refer to “[2. Basic Python Service](#_Project_Launch_Recommendations:)” for details on installation and first launch.