UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERIA

ESCUELA DE CIENCIAS Y SISTEMAS

INTRODUCCIÓN A LA PROGRAMACIÓN Y COMPUTACIÓN 1

SECCIÓN E

PRIMER SEMESTRE 2022

ING. NEFTALÍ DE JESÚS ALDANA CALDERÓN

AUX. SERGIO FERNANDO OTZOY GONZALEZ

SEGUNDO PROYECTO DE LABORATORIO

BIBLIOTECA – API

CONTENIDO

Objetivos 2

General 2

Específicos 2

Descripción 2

Requisitos funcionales 3

Manejo de libros 3

Crear libro 3

Actualizar información de un libro 4

Buscar un libro 4

Préstamos y devoluciones 5

Crear prestamista 5

Consultar información de prestamista 6

Prestar libro 7

Devolver libro 8

Despliegue 9

Documentación 10

Repositorio github 11

Consideraciones finales 11

Requerimientos para el desarrollo del proyecto 12

OBJETIVOS

GENERAL

- Usar algoritmos y programación orientada a objetos para desarrollar una API con el lenguaje de programación Python

ESPECÍFICOS

- Usar el Framework Flask de Python para desarrollar una API.
- Usar Heroku como proveedor Cloud (PaaS) para desplegar una API.

DESCRIPCIÓN

Anteriormente, el administrador (a punto de jubilarse) de una pequeña biblioteca comunitaria solicitó una aplicación para simplificar el manejo de dicha biblioteca. Esa aplicación fue presentada al administrador y resultó de su agrado.

El nuevo administrador tomó control de la biblioteca y se ha dado cuenta que una aplicación de escritorio no es suficiente para cubrir todas las necesidades de la biblioteca. El nuevo administrador es visionario y cree que la antigua forma de administrar no le asegurará una larga vida a la biblioteca. Ha solicitado la creación de una API desde la cuál otras personas o dominios web puedan acceder a los servicios de la biblioteca.

El nuevo proyecto es ambicioso y tomará varios meses antes de que sea completamente funcional. Se solicita que cree una primera versión (prototipo) del API de la biblioteca.

REQUISITOS FUNCIONALES

A continuación, se presentan los contratos de los diferentes servicios que deberá cumplir la API. Estos contratos se establecieron con el objetivo de estandarizar la comunicación con otros servicios, páginas web, aplicaciones móviles, etc.

Si en la petición a un servicio (crear libro, consultar libro, etc) no existe ningún error, la respuesta de esa petición deberá contener el código http 200 y un mensaje que informe al usuario del éxito de la operación (donde sea necesario). Si existe algún error, ya sea porque la petición está mal hecha, falta algún campo, etc., el código http a usar deberá estar en el rango 4xx~5xx y un mensaje que informe del fracaso de la operación.

MANEJO DE LIBROS

CREAR LIBRO

Añadirá un nuevo registro de un libro a la biblioteca. Deberá realizar la siguiente verificación:

- No debe existir ISBN duplicados. Si se intenta crear un libro con ISBN la respuesta contendrá un mensaje de error.

METODO	POST
ENDPOINT	/book
REQUEST (BODY)	`{ "isbn": number, "author": string, "title": string, "year": number, "no_copies": number, "no_available_copies": number, }`
RESPONSE	HTTP CODE 200 y 4xx~5xx: `{ "msg": string }`

ACTUALIZAR INFORMACIÓN DE UN LIBRO

Actualizará la información de un libro reemplazándola. Es decir, los datos que sean usados en la petición reemplazarán los registros actuales del libro a excepción del isbn, las copias disponibles y copias disponibles actuales. Se deberá realizar la siguiente validación:

- Que el ISBN exista. Si el ISBN no existe la respuesta contendrá un mensaje de error.

METODO	PUT
ENDPOINT	/book
REQUEST (BODY)	`{ "isbn": number, "author": string, "title": string, "year": number }`
RESPONSE	HTTP CODE 200 y 4xx~5xx: `{ "msg": string }`

BUSCAR UN LIBRO

Para este servicio no habrá excepciones (errores) si no variaciones. Se podrá realizar tres tipos de búsqueda:

- Por título
- Por rango de año
- Por nombre de autor

Esto quiere decir que, al realizar la petición puede existir el argumento del título o no, puede existir el rango del año o no, puede existir el nombre del autor o no.

La respuesta de este servicio variará de la siguiente forma:

- Si no hay libros que coincidan con los criterios de búsqueda, la respuesta deberá estar “vacía”.
- Si la búsqueda se realiza sin ningún criterio, la respuesta deberá contener todos los libros de la biblioteca

METODO	GET
ENDPOINT	/book
REQUEST (QUERY)	`{ "title": string, "year_from": number, "year_to": number, "author": string }`

RESPONSE	HTTP CODE 200:
	`[ { "isbn": number, "author": string, "title": string, "year": number, "no_copies": number, "no_available_copies": number, } ]`
	Si no hay coincidencias la respuesta seguirá siendo con HTTP CODE 200 pero deberá ser un arreglo vacío:
	`[ ]`
	HTTP CODE 4xx~5xx:
	`{ "msg": string }`

PRÉSTAMOS Y DEVOLUCIONES

CREAR PRESTAMISTA

Creará un nuevo prestamista. Se deberá realizar las siguientes verificaciones:

- Si el CUI ya existe, la respuesta contendrá un mensaje de error.

METODO	POST
ENDPOINT	/person
REQUEST (BODY)	`{ "cui": string, "last_name": string, "first_name": string }`
RESPONSE	HTTP CODE 200 y 4xx~5xx: `{ "msg": string }`

CONSULTAR INFORMACIÓN DE PRESTAMISTA

Para este servicio se usará el CUI de la persona para buscar su información. Si el CUI si existe, la respuesta contendrá la información personal y el historial de préstamos de la persona. En caso no haya prestado ningún libro aún, el servicio deberá devolver un arreglo vacío.

Se deberá realizar la siguiente verificación:

- Si el CUI no existe, la respuesta contendrá un mensaje de error.

METODO	GET
ENDPOINT	/person/:cui
REQUEST (PARAMS)	`{ "cui": string }`
RESPONSE	HTTP CODE 200:

```
{
    "cui": string,
    "first_name": string,
    "last_name": string,
    "record": [
        {
            "uuid": string,
            "isbn": number,
            "title": string,
```

	<pre>"lend_date ": date,
	"return_date": date
	}
	]
	}</pre>
	Si la persona no ha realizado prestamos:
	<pre>{
	"cui": string,
	"first_name": string,
	"last_name": string,
	"record": []
	}</pre>
	HTTP CODE 4xx~5xx:
	<pre>{
	"msg": string
	}</pre>

PRESTAR LIBRO

Prestará un libro a un prestamista registrado previamente en el sistema. Deberá realizar las siguientes verificaciones:

- El CUI del prestamista debe existir.
- El ISBN del libro debe existir
- El prestamista con el CUI asociado **no** debe tener algún libro pendiente de regresar

Si no se cumple cualquiera de las verificaciones la respuesta contendrá un mensaje de error

METODO	POST
ENDPOINT	/borrow

REQUEST (BODY)	<pre>{
	"cui": string,
	"isbn": number
	}</pre>

RESPONSE	HTTP CODE 200: <pre>{</pre>

	<pre>{
	"uuid": string
	}</pre> HTTP CODE 4xx~5xx: <pre>{
	"msg": string
	}</pre>

Para este servicio, internamente en la API, deberá generar un identificador único universal (UUID) para cada préstamo realizado. Ese identificador servirá posteriormente para devolver el libro. Además de esto, se deberá almacenar la fecha y hora de préstamo, el ISBN y el título del libro. La *cantidad de copias disponibles actualmente* deberá disminuir al prestar un libro exitosamente.

DEVOLVER LIBRO

Devolverá un libro que un prestamista haya prestado anteriormente. Para devolver el libro se usará el identificar único universal que se generó al prestar el libro. Deberá realizar la siguiente verificación:

- El UUID deberá existir, sino la respuesta contendrá un mensaje de error.

METODO	PATCH
ENDPOINT	/borrow/:uuid

REQUEST (PARAMS)	<pre>{
	"uuid": string,
	}</pre>

RESPONSE	HTTP CODE 200 y 4xx~5xx: <pre>{
	"msg": string
	}</pre>

Actualizará el registro del préstamo que se generó al usar el servicio de *prestar libro* agregando la fecha y hora de devolución. La *cantidad de copias disponibles actualmente* deberá aumentar al devolver un libro exitosamente.

DESPLIEGUE

Para desplegar la API deberá hacer uso del proveedor Cloud **Heroku**.

Deberá crear una aplicación con el nombre ipc1-proyecto2-*carnet*. La región por usar será *United States*

Ilustración 1. Crear nueva aplicación en el dashboard de Heroku

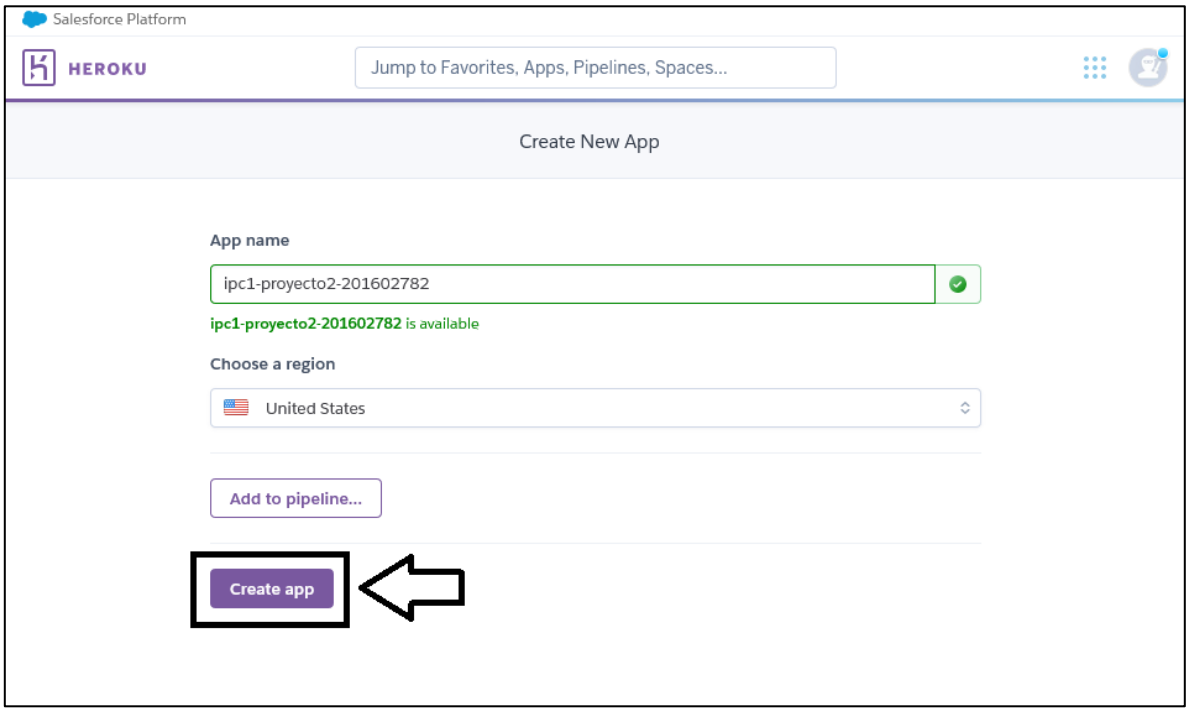

Ilustración 2. Especificando el nombre de una nueva aplicación en Heroku

DOCUMENTACIÓN

Se deberá documentar todos los servicios de la biblioteca usando Postman. La documentación deberá contar con ejemplos que muestren como varían las repuestas de los diferentes servicios en diferentes escenarios. Esta documentación deberá estar publicada y disponible a través de la web.

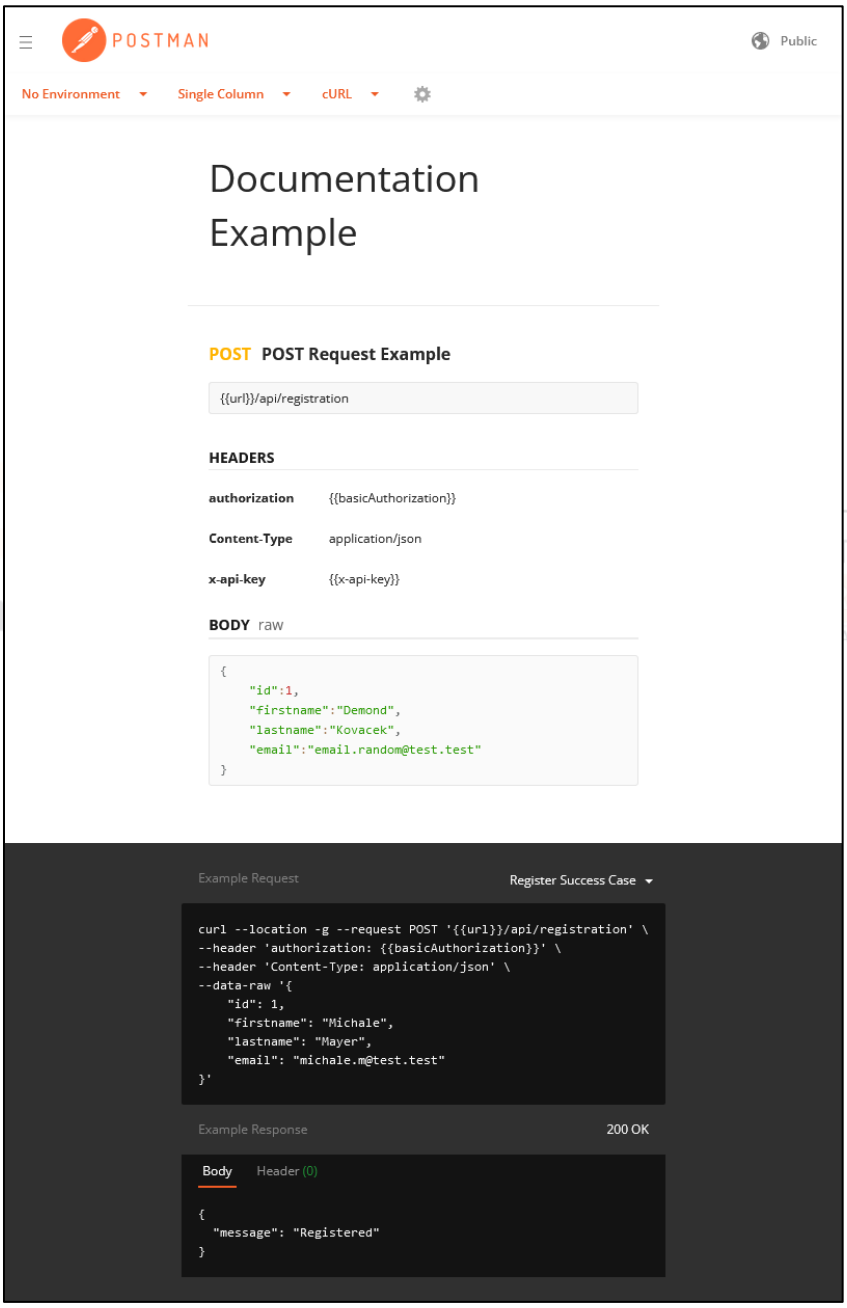

Ilustración 3. Ejemplo de documentación en Postman

REPOSITORIO GITHUB

Se usará GitHub como control de versionamiento. Todo el código fuente deberá estar en un repositorio en GitHub:

1. El repositorio debe ser **privado.**
2. El nombre del repositorio será **ipc1-1s2022-proyecto2.**
3. Deberá añadir al usuario **Otzoy97** como colaborador.
4. Deberá crear un archivo README.md en la raíz del proyecto. Dicho archivo contendrá los siguientes datos:
    a. Identificación del curso (nombre del curso, sección, etc).
    b. Identificación del estudiante (carné, nombre completo).
    c. Enlace de documentación en Postman.

CONSIDERACIONES FINALES

1. Toda la información que maneje la API se manejará en memoria usando arreglos.
   Si el estudiante así lo desea, puede hacer uso de alguna base de datos SQL o No-SQL. **No es obligatorio implementar una base de datos**.
2. La calificación consistirá en conectar su API a una página web que consumirá los servicios, por lo que es importante que cumpla con los contratos que se describen en el enunciado. **No deberá construir una página web**.
3. Toda la aplicación deberá estar disponible a través del servicio que ofrece Heroku. Cualquier aplicación que solo esté disponible de forma local tendrá penalización durante la calificación.
4. Todo el código fuente deberá estar en un repositorio en GitHub
    a. El repositorio debe ser **privado.**
    b. El nombre del repositorio será **ipc1-1s2022-proyecto2.**
    c. Deberá añadir al usuario **Otzoy97** como colaborador.

REQUERIMIENTOS PARA EL DESARROLLO DEL PROYECTO

- Documentación
  1. La documentación deberá estar publicada en la web, disponible a través de los servicios que ofrece Postman.
     a. El enlace de la documentación deberá estar en el archivo README.md del repositorio de GitHub.
- Restricciones y penalizaciones
  1. La aplicación deberá ser desarrollada completamente en Python usando el Framework Flask.
  2. Se calificará desde la aplicación desplegada en Heroku. De lo contrario se aplicará una penalización.
  3. Cualquier servicio implementado que no cumpla con los contratos tendrá penalización.
  4. El IDE por utilizar queda a discreción del estudiante.
  5. Copias obtendrán nota 0 y serán reportadas a la Escuela de Ciencias y Sistemas.
- Entregables
  1. **Fecha de entrega**: 24/04/2022 antes de las 23:59 PM. No se aceptarán entregas luego de esa hora. No se aceptarán entregas por correo.
  2. Entregar únicamente el enlace de su repositorio de GitHub en el entregable en UEDI. Se verificará que el repositorio no tenga actividades ni commits luego de la fecha de entrega. **Si hay evidencia de modificación de *código* se realizará un rollback y desplegará la versión anterior más cercana a la fecha y hora de entrega**.