# Learning Objective

Create a Book service using Flask and SQL Alchemy.

# Introduction

*We shall create a Book service with 3 operations:*

* *Get all books*
* *Find book by isbn13*
* *Create book*

# Preparation

1. Use *book.sql* to create and populate the database in WampServer
2. Using the command prompt, create a project folder named **book** in C:\ESD\Labs:

**mkdir C:\ESD\Labs\book**

1. Open the project folder in VS Code and create a terminal:

**cd C:\ESD\Labs\book**

**code .**

**Terminal → New Terminal**

# 

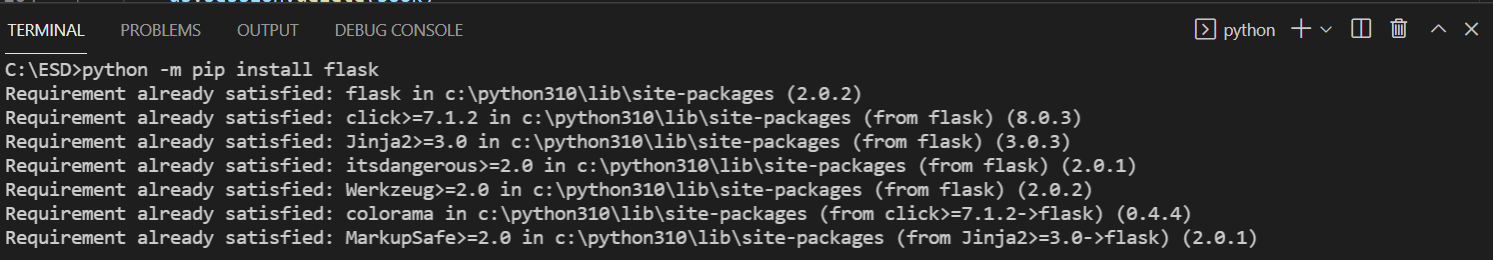
# Create the routes (endpoints)

*We could use Flask extensions such as* [*Flask-RESTFlask*](https://flask-restplus.readthedocs.io/en/stable/)*,* [*Flask-RESTful*](https://flask-restful.readthedocs.io/en/latest/) *or* [*Connexion*](https://connexion.readthedocs.io/en/latest/) *to make it easier to build RESTful APIs but to keep things simple, we shall just use basic Flask.*

1. Use the following command in the VS Code terminal to install Flask, if you have not already done so:

**python -m pip install flask**

Otherwise, it may report that the requirement is already satisfied:



1. In VS Code, create a new file in your project folder named **book.py**
2. In book.py, add the following code:

from flask import Flask, request, jsonify

app = Flask(\_\_name\_\_)

@app.route("/book")

def get\_all():

pass

@app.route("/book/<string:isbn13>")

def find\_by\_isbn13(isbn13):

pass

@app.route("/book/<string:isbn13>", methods=['POST'])

def create\_book(isbn13):

pass

if \_\_name\_\_ == '\_\_main\_\_':

app.run(port=5000, debug=True)

Explanation

from flask import Flask, request, jsonify

app = Flask(\_\_name\_\_)

*Import Flask and initialize a Flask application.*

*We also import request and jsonify which we will need to use later.*

@app.route("/book")

def get\_all():

pass

*Use Flask's app.route decorator to map the URL route /book to the function get\_all.*

*To call this function, the URL to use is for e.g. GET http://127.0.0.1:5000/book. Currently, the function does not do anything yet.*

@app.route("/book/<string:isbn13>")

def find\_by\_isbn13(isbn13):

pass

*Use Flask's app.route decorator to map the URL route /book/isbn13 to the function find\_by\_isbn13. isbn13 is a path variable of string type.*

*To call this function, the URL to use is for e.g. GET http://127.0.0.1:5000/book/1234567890123.*

*The function will have access to the isbn13 variable.*

*Currently, the function does not do anything yet.*

@app.route("/book/<string:isbn13>", methods=['POST'])

def create\_book(isbn13):

pass

*Use Flask's app.route decorator to map the URL route /book/isbn13 to the function create\_book. isbn13 is a path variable of string type.*

*By default, @app.route uses a GET method. If we want to use any other HTTP methods, we have to specify them by passing them via the methods parameter as a list.*

*To call this function, the URL to use is for e.g. POST http://127.0.0.1:5000/book/1234567890123. The details of the book have to be sent in the body of the request.*

*Currently, the function does not do anything yet.*

if \_\_name\_\_ == '\_\_main\_\_':

app.run(port=5000, debug=True)

*We run our application behind an if guard. This will ensure that we don't start up the web server if we import this script into another one (we'll only run the web server if we run this file explicitly).*

*In addition, we specify the port to use is 5000 (which is the default port anyway) and set debug to True, which will provide debugging information and also restart the app automatically if the code is modified while the app is running.*

# 

# Use Flask-SQLAlchemy

# *Ref: REST APIs with Flask and Python - Jose Salvatierra (Chapter 6: Simplifying storage with Flask-SQLAlchemy)*

# [*https://learning-oreilly-com.libproxy.smu.edu.sg/videos/rest-apis-with/9781788621526/9781788621526-video6\_1*](https://learning-oreilly-com.libproxy.smu.edu.sg/videos/rest-apis-with/9781788621526/9781788621526-video6_1)

*The app needs to interact with the book database to get all books, find book by isbn13 or create book.*

*Instead of using raw SQL, we shall use Flask-SQLAlchemy, which is a Python SQL toolkit and Object Relational Mapper (ORM).*

*ORM can be thought of as a translator that translates Python code to SQL. Hence, SQLAlchemy can be used to easily store objects into a relational database.*

1. Use the following command in the VS Code terminal to install Flask-SQLAlchemy:

**python -m pip install Flask-SQLAlchemy**

*Note: We use Flask’s version of SQLAlchemy which relies on SQLAlchemy. It*

*aims to simplify using SQLAlchemy with Flask.*

1. Use the following command in the VS Code terminal to install MySQL Python Connector:

**python -m pip install mysql-connector-python**

1. In book.py, add the following code

(on line 2 after the Flask import statement):

from flask import Flask, request, jsonify

from flask\_sqlalchemy import SQLAlchemy

(on line 5 after the Flask initialization statement):

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql+mysqlconnector://root@localhost:3306/book'

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

app.config['SQLALCHEMY\_ENGINE\_OPTIONS'] = {'pool\_recycle': 299}

db = SQLAlchemy(app)

class Book(db.Model):

\_\_tablename\_\_ = 'book'

isbn13 = db.Column(db.String(13), primary\_key=True)

title = db.Column(db.String(64), nullable=False)

price = db.Column(db.Float(precision=2), nullable=False)

availability = db.Column(db.Integer)

def \_\_init\_\_(self, isbn13, title, price, availability):

self.isbn13 = isbn13

self.title = title

self.price = price

self.availability = availability

def json(self):

return {"isbn13": self.isbn13, "title": self.title, "price": self.price, "availability": self.availability}

Explanation

from flask\_sqlalchemy import SQLAlchemy

*Import Flask’s version of SQLAlchemy, which implicitly extends and relies on the base installation of SQLAlchemy.*

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql+mysqlconnector://root:@localhost:3306/book'

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False  
app.config['SQLALCHEMY\_ENGINE\_OPTIONS'] = {'pool\_recycle': 299}

*The SQLAlchemy Database URI format is:* ***dialect+driver://username:password@host:port/database***

*For MySQL, the value of* ***dialect+driver*** *is* ***mysql+mysqlconnector****.*

*Using the above line of code, we specify the database URL. Here we use the mysql+mysqlconnector prefix to tell SQLAlchemy which database engine and connector we are using. And here we assume no password is required for the root user to access the database, hence no password is specified in the database URI.*

*If the database requires a password to access, the URI should include the password value. For example, if the username and password are* ***root*** *and* ***root****, the above URI setting code will change as:*

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql+mysqlconnector://root:root@localhost:3306/book'

*For MacOS users, if you use MAMP, it is likely you have to use the above URI with password* ***root****.*

*We also disable modification tracking as it requires extra memory and is not needed.*

*MySQL has an automatic connection close behavior, for connections that have been idle for a fixed period of time, defaulting to eight hours. To circumvent having timeout issue, we set the pool\_recycle option to discard and replace a connection if it has been present in the pool for 299 seconds:*

db = SQLAlchemy(app)

*We initialize a connection to the database and keep this in the db variable and use this to interact with the database.*

class Book(db.Model):

*We create a new class Book, which inherits from a basic database model, provided by SQLAlchemy. This will also make SQLAlchemy create a table called book (****if*** *we run the db.create\_all() function), which it will use to store our Book objects.*

\_\_tablename\_\_ = 'book'

isbn13 = db.Column(db.String(13), primary\_key=True)

title = db.Column(db.String(64), nullable=False)

price = db.Column(db.Float(precision=2), nullable=False)

availability = db.Column(db.Integer)

*However, we explicitly specify the table name as book (so it is possible to*

*create or use an existing table with a different name from the class).*

*We specify attributes of our book called isbn13, title, price and availability. SQLAlchemy will use these as column names in the table. In the case where you are using an existing table, we use this to tell SQLAlchemy about our table and columns. You should already be familiar with the other parameters, like data type, primary key, nullable, etc.*

def \_\_init\_\_(self, isbn13, title, price, availability):

self.isbn13 = isbn13

self.title = title

self.price = price

self.availability = availability

*We specify the properties of a Book when it is created.*

def json(self):

return {"isbn13": self.isbn13, "title": self.title, "price": self.price, "availability": self.availability}

*We specify how to represent our book object as a JSON string.*

Now, we can define the code in each function.

1. In book.py, replace *pass* with the following code within the function *get\_all*:

@app.route("/book")

def get\_all():

booklist = db.session.scalars(db.select(Book)).all()

if len(booklist):

return jsonify(

{

"code": 200,

"data": {

"books": [book.json() for book in booklist]

}

}

)

return jsonify(

{

"code": 404,

"message": "There are no books."

}

), 404

Explanation

booklist = db.session.scalars(db.select(Book)).all()

*SQLAlchemy provides a session.scalars attribute to retrieve all records from the book table using db.select(Book).all(); this returns a list, which we assign to booklist.*

if len(booklist):

return jsonify(

{

"code": 200,

"data": {

"books": [book.json() for book in booklist]

}

}

)

*If the book list is not empty, we use for book to perform an iteration and create a JSON representation of it using book.json() function. We then return the code (which corresponds to the HTTP status) and list of books in the JSON representation using jsonify. See the Book API (link in the Reference section) for the response format.*

*Note: In order to have access to the jsonify object, we need to import it from Flask, which we already did earlier (on the 1st line of code):*

*from flask import Flask, request, jsonify*

return jsonify(

{

"code": 404,

"message": "There are no books."

}

), 404

*Else, return an error message in JSON. In addition, return HTTP status code 404 for NOT FOUND. If not specified, 200 OK will be returned.*

1. In book.py, replace *pass* with the following code within the function *find\_by\_isbn13*:

@app.route("/book/<string:isbn13>")

def find\_by\_isbn13(isbn13):

book = db.session.scalar(

db.select(Book).filter\_by(isbn13=isbn13)

)

if book:

return jsonify(

{

"code": 200,

"data": book.json()

}

)

return jsonify(

{

"code": 404,

"message": "Book not found."

}

), 404

Explanation

book = db.session.scalar(

db.select(Book).filter\_by(isbn13=isbn13)

)

*We use db.session.scalar(db.select(Book).filter\_by(isbn13=isbn13))*

*to retrieve only the book with the isbn13 specified in the path variable[[1]](#footnote-0). This is similar to the WHERE clause in a SQL SELECT expression.   
This will return 1 book or None if there is no matching book.*

if book:

return jsonify(

{

"code": 200,

"data": book.json()

}

)

return jsonify(

{

"code": 404,

"message": "Book not found."

}

), 404

*If the book is found (not None), return its JSON representation.*

*Else, return an error message in JSON. In addition, return HTTP status code 404 for NOT FOUND. If not specified, 200 OK will be returned.*

1. In book.py, replace *pass* with the following code within the function *create\_book*:

@app.route("/book/<string:isbn13>", methods=['POST'])

def create\_book(isbn13):

if db.session.scalar(db.select(Book).filter\_by(isbn13=isbn13)):

return jsonify(

{

"code": 400,

"data": {

"isbn13": isbn13

},

"message": "Book already exists."

}

), 400

data = request.get\_json()

book = Book(isbn13, \*\*data)

try:

db.session.add(book)

db.session.commit()

except:

return jsonify(

{

"code": 500,

"data": {

"isbn13": isbn13

},

"message": "An error occurred creating the book."

}

), 500

return jsonify(

{

"code": 201,

"data": book.json()

}

), 201

Explanation

if db.session.scalar(db.select(Book).filter\_by(isbn13=isbn13)):

return jsonify(

{

"code": 400,

"data": {

"isbn13": isbn13

},

"message": "Book already exists."

}

), 400

*First, we use db.session.scalar(db.select(Book).filter\_by(isbn13=isbn13)) to check if the book already exists in the table. If it does, return an error message in JSON with HTTP status code 400 BAD REQUEST.*

data = request.get\_json()

book = Book(isbn13, \*\*data)

*The details of the book have to be sent in the body of the request in JSON format. Hence, we use get\_json() to get the data from the request received.*

*Then we create an instance of a book using the isbn13 and the attributes sent in the request (\*\*data). \*\* is a common idiom to allow an arbitrary number of arguments to a function, in this case, all attributes retrieved from request, instead of individually specifying each.*

*Note: In order to have access to the request object, we need to import it from Flask, which we already did earlier (on the 1st line of code):*

*from flask import Flask, request, jsonify*

try:

db.session.add(book)

db.session.commit()

except:

return jsonify(

{

"code": 500,

"data": {

"isbn13": isbn13

},

"message": "An error occurred creating the book."

}

), 500

*To add the book to the table and commit the changes, we use the db.session object (provided by SQLAlchemy) - the current database connection.*

*We do this in the try-except block and return an error message in JSON with HTTP status code 500 - INTERNAL SERVER ERROR if an exception occurs.*

return jsonify(

{

"code": 201,

"data": book.json()

}

), 201

*Finally, if there is no exception, we return the JSON representation of the book we have added with HTTP status code 201 - CREATED.*

1. Save the file.

Now, we are ready to test the service.

# Test the service

1. *Start Wamp server and use book.sql to create and populate the*

*database*

1. Download and install the [*Postman app*](https://www.getpostman.com/)- we shall use this for testing the service instead of the browser
2. In the VS Code terminal, run the app by entering:

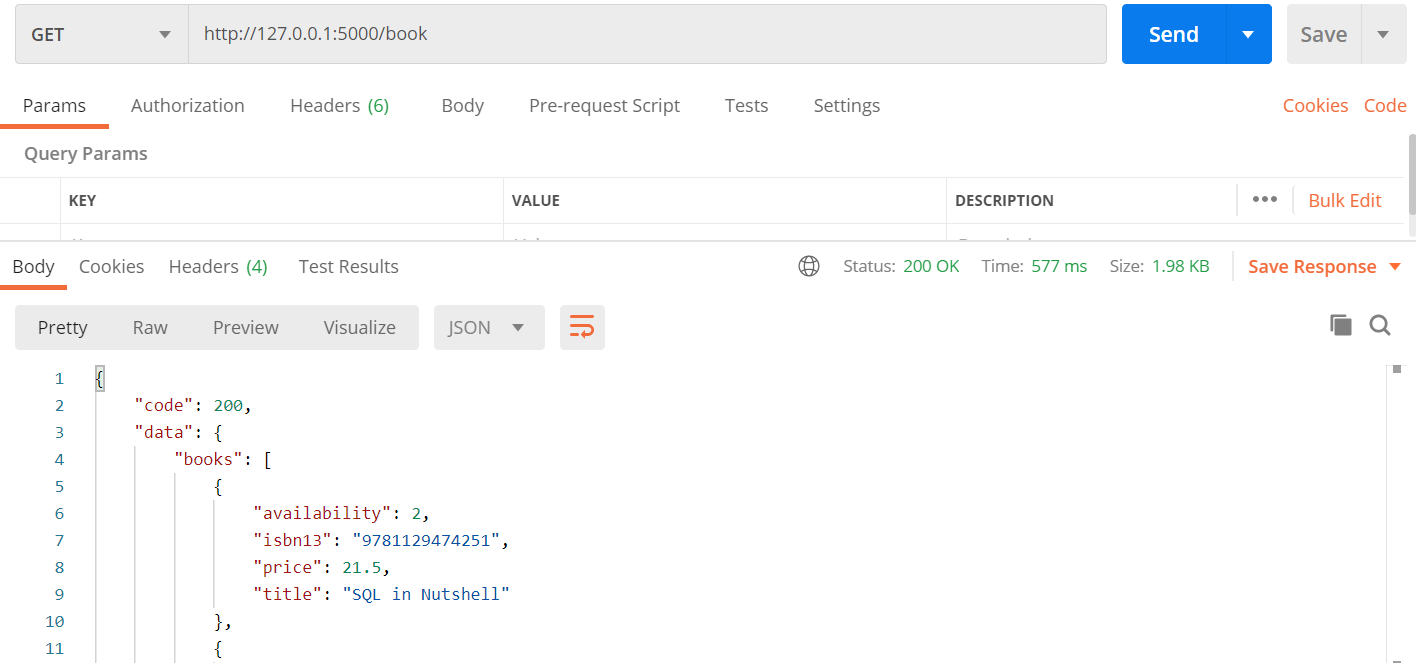
python book.py

*Note: Because, we have added the following, we can run it as above instead of* **python -m flask run***(which we used in the earlier lab)*

if \_\_name\_\_ == '\_\_main\_\_':

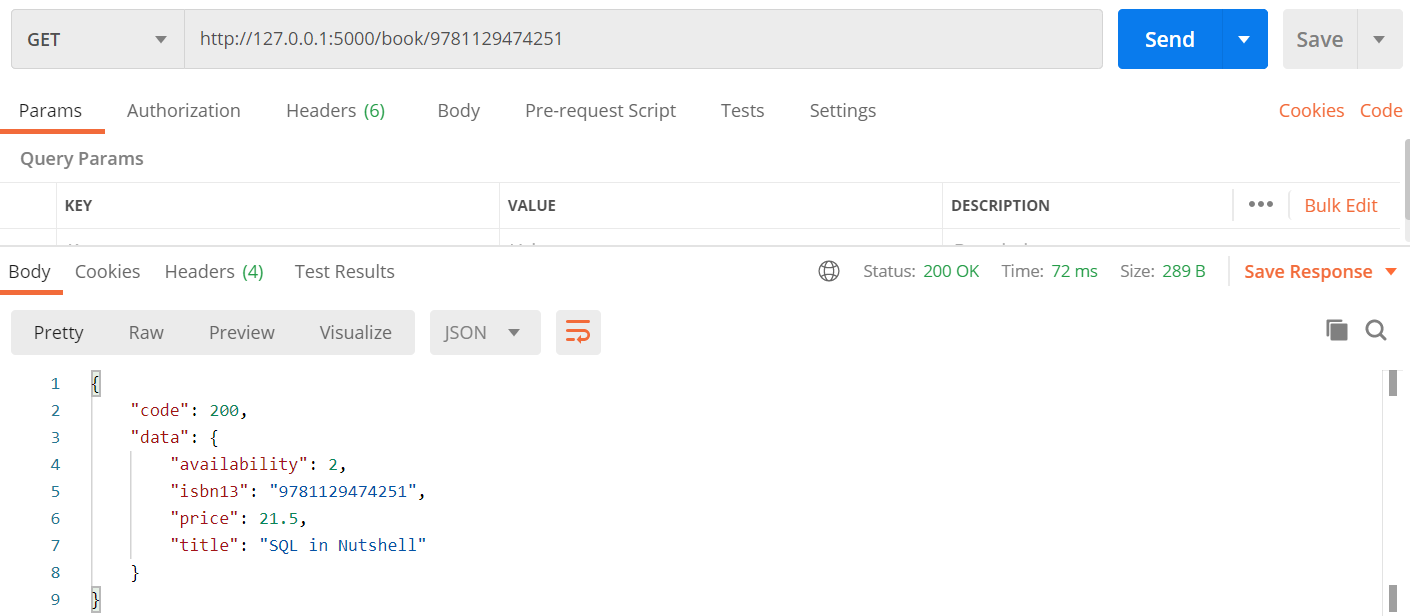
app.run(port=5000, debug=True)

1. Launch the Postman app
2. Click  to open a request tab if one is not already opened
3. Enter [*http://127.0.0.1:5000/book*](http://127.0.0.1:5000/book) into the URL field and click **Send**. You should see the list of books:



The status should be **200 OK**.

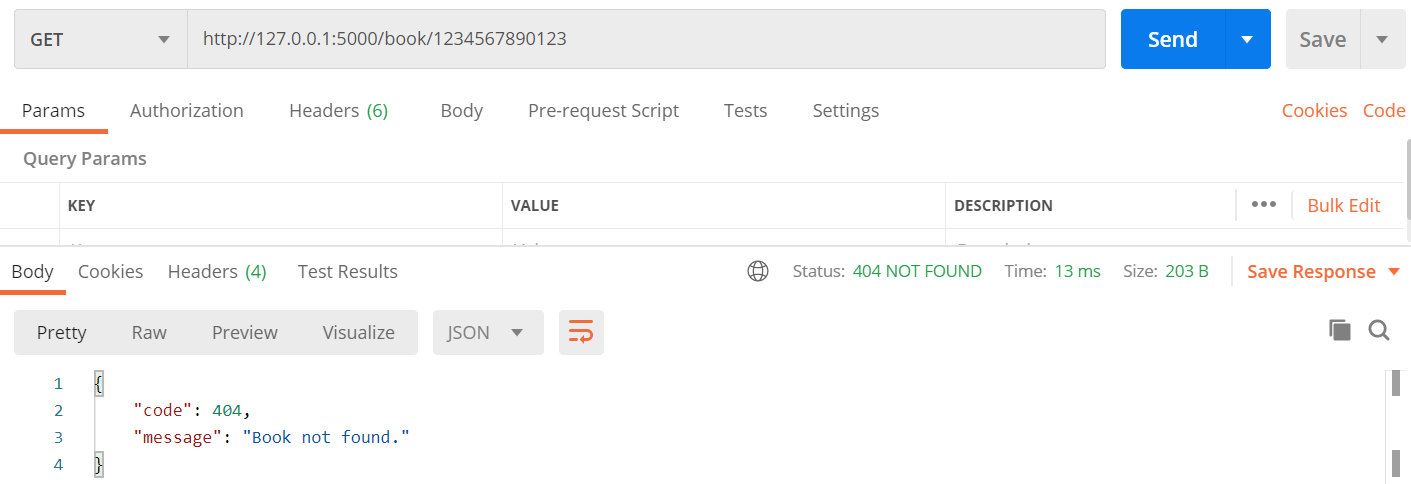
1. Click  to open another request tab and enter [*http://127.0.0.1:5000/book/9781129474251*](http://127.0.0.1:5000/book/9781129474251) into the URL field and click **Send**. You should see the book with the corresponding isbn13:



The status should be **200 OK**.

1. Change the isbn13 to 1234567890123 and click **Send**.

You should see an error message:



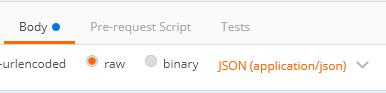
The status should be 404 NOT FOUND.

1. Click  to open another request tab.

Change the method to **POST** and enter [*http://127.0.0.1:5000/book/9781449474453*](http://127.0.0.1:5000/book/9781449474453)into the URL field.

In the *Body* tab,   
I. select **raw** for format and

II. choose **JSON (application/json)**



Then, enter the following:

{

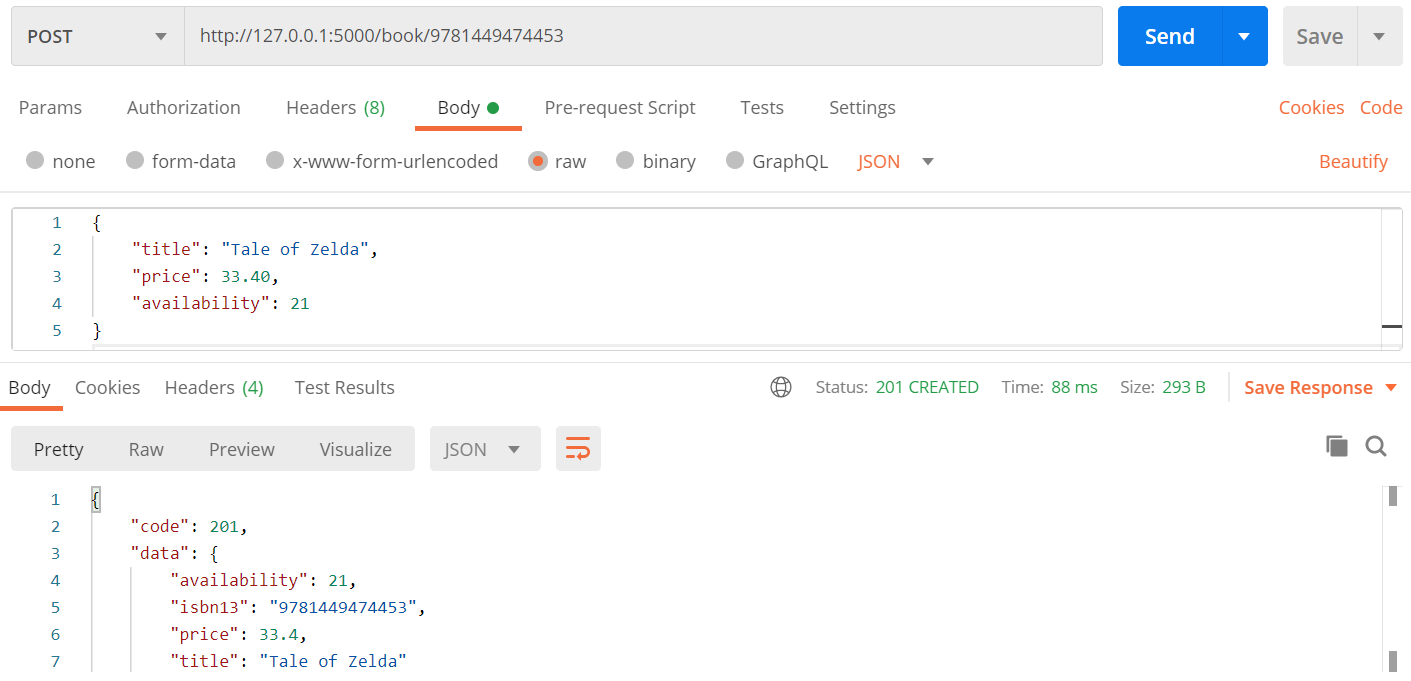
"title": "Tale of Zelda",

"price": 33.40,

"availability": 21

}

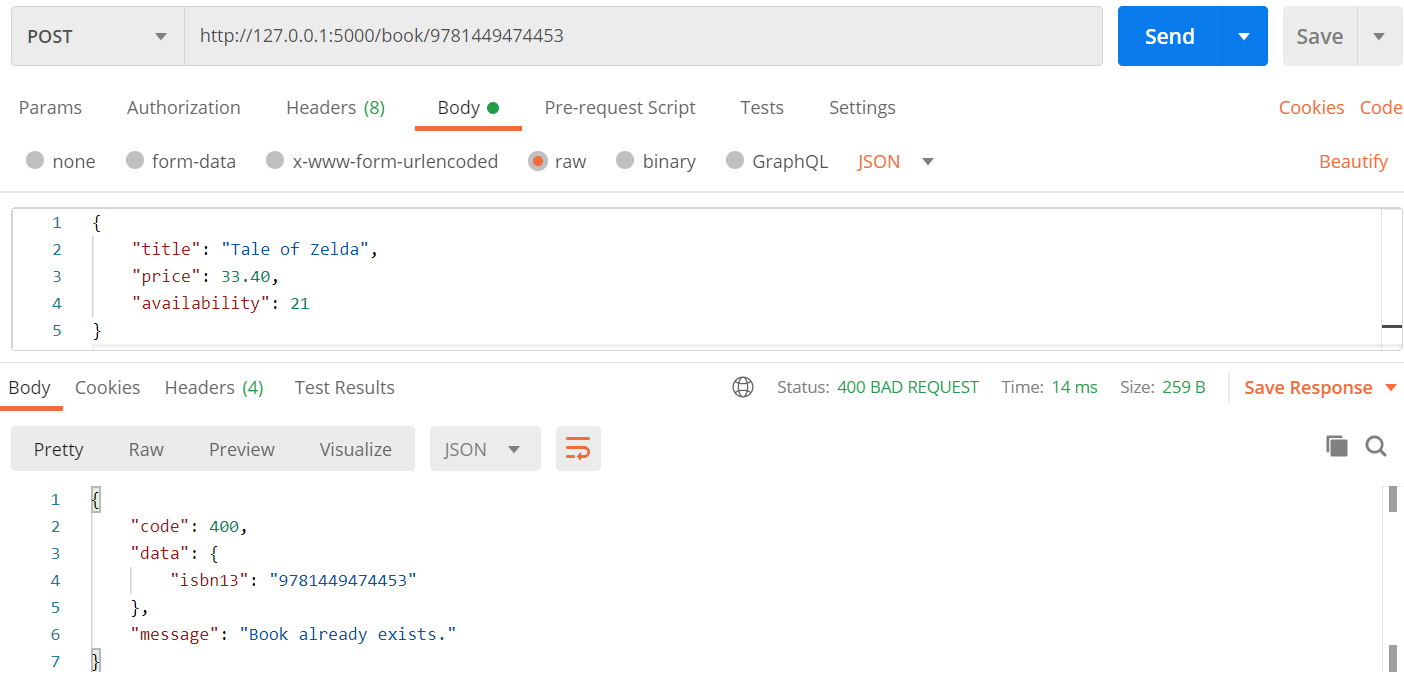
Click **Send[[2]](#footnote-1)**. You should see the book details corresponding to your input:



The status should be 201 CREATED.

Check your database table to verify that the book was added.

1. Now, click **Send** again. You should see an error message:



The status should be 400 BAD REQUEST.

# Learning Points

* *Create a microservice using Python Flask (web framework), SQLAlchemy, WampServer (MySQL)*
* *Using variables in app.route decorator(s)*
* *Creating service operations*
* *Using Flask SQLAlchemy to interact with a database like MySQL*
* *Testing the Book service using a test tool such as Postman app*

# Reference

1. [*Book API*](https://docs.google.com/document/d/15f-zExUvs70orOFUGJkcCybSJjIGgjlKdDv-nBPFm-Y/edit?usp=sharing)

# *book.py*

from flask import Flask, request, jsonify

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql+mysqlconnector://root@localhost:3306/book'

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

app.config['SQLALCHEMY\_ENGINE\_OPTIONS'] = {'pool\_recycle': 299}

db = SQLAlchemy(app)

class Book(db.Model):

\_\_tablename\_\_ = 'book'

isbn13 = db.Column(db.String(13), primary\_key=True)

title = db.Column(db.String(64), nullable=False)

price = db.Column(db.Float(precision=2), nullable=False)

availability = db.Column(db.Integer)

def \_\_init\_\_(self, isbn13, title, price, availability):

self.isbn13 = isbn13

self.title = title

self.price = price

self.availability = availability

def json(self):

return {"isbn13": self.isbn13, "title": self.title, "price": self.price, "availability": self.availability}

@app.route("/book")

def get\_all():

booklist = db.session.scalars(db.select(Book)).all()

if len(booklist):

return jsonify(

{

"code": 200,

"data": {

"books": [book.json() for book in booklist]

}

}

)

return jsonify(

{

"code": 404,

"message": "There are no books."

}

), 404

@app.route("/book/<string:isbn13>")

def find\_by\_isbn13(isbn13):

book = db.session.scalar( db.select(Book).filter\_by(isbn13=isbn13) )

if book:

return jsonify(

{

"code": 200,

"data": book.json()

}

)

return jsonify(

{

"code": 404,

"message": "Book not found."

}

), 404

@app.route("/book/<string:isbn13>", methods=['POST'])

def create\_book(isbn13):

if db.session.scalar( db.select(Book).filter\_by(isbn13=isbn13) ):

return jsonify(

{

"code": 400,

"data": {

"isbn13": isbn13

},

"message": "Book already exists."

}

), 400

data = request.get\_json()

book = Book(isbn13, \*\*data)

try:

db.session.add(book)

db.session.commit()

except:

return jsonify(

{

"code": 500,

"data": {

"isbn13": isbn13

},

"message": "An error occurred creating the book."

}

), 500

return jsonify(

{

"code": 201,

"data": book.json()

}

), 201

if \_\_name\_\_ == '\_\_main\_\_':

app.run(port=5000, debug=True)

# Troubleshooting

### 1.Test Create book using Postman fails

| *Error:* Error is: TypeError: DefaultMeta object argument after ** must be a mapping, not NoneType more information omitted |
| --- |
| *Reason:* The content-type of this request should be set to *application/json*. By selecting raw and JSON (in the dropdown list), it should automatically do this. Some Postman versions (e.g. v7.9.0) do not do this automatically.  This error is thrown as the line  data = request.get\_json()  will return None, thus data becomes None since the data returned is not expected to be of the content-type application/json.  Thus when we try to do this with \*\*data, it could not reference the data from None.  book = Book(isbn13, \*\*data) |
| *How to fix:*   1. Upgrade Postman to at least v7.14 2. Create a new post request (DO NOT REUSE THE OLD ONE as it will continue to act weird) 3. And try to Send the request again. |

### 2. *VSC error: Instance of SQLAlchemy has no column membe*r

| *Error:*  The “db” are underlined with a red twirling line, and there is an error message “Instance of SQLAlchemy has no column member” |
| --- |
| *How to fix:*  *Reference:* [*https://stackoverflow.com/questions/53975234/instance-of-sqlalchemy-has-no-column-member*](https://stackoverflow.com/questions/53975234/instance-of-sqlalchemy-has-no-column-member)   1. **python -m pip install pylint-flask** 2. **python -m pip install pylint-flask-sqlalchemy** 3. Go to Visual Studio Code settings 4. See screenshot below    1. Search for *python.linting.pylintArgs*    2. Add the following items:       1. --load-plugins       2. pylint-flask       3. pylint-flask-sqlalchemy |

### 3. *SQLAlchemy Error - Character set ‘255’ unsupported*

| *Error:* |
| --- |
| *Reason: sqlalchemy.exc.ProgrammingError: (mysql.connector.errors.ProgrammingError) Character set '255' unsupported*  mysql-connector might not be well supported by sqlalchemy |
| *How to fix:*   1. Install pymysql: **python -m pip install pymysql** 2. Change the SQALCHEMY\_DATABASE\_URI to   app.config['SQLALCHEMY\_DATABASE\_URI'] = 'mysql+pymysql://root@localhost:3306/book' |

### *4. MacOS > MAMP > Unable to start Apache on port 80 and there is no other process using it.*

| *Error:* |
| --- |
| *Reason:* Root-level access right is required to start apache on ports 1024 and lower. No idea why. |
| *How to fix:*   1. Launch MAMP. 2. Click CMD+comma and click "Ports". Select 80 & 3306. 3. On the main MAMP screen, click "Start". The Apache web server will fail to start, but the MySQL server will launch correctly. 4. Open up a Terminal (use spotlight then search "Terminal") 5. Enter:   sudo /Applications/MAMP/Library/bin/apachectl start  (you will be asked to enter your Mac's root password) 6. Voila! Everything should be working.   *Alternative solution*  Run MAMP apache on port 8888 and MySQL on port 8889.  This comes with caveat that you will need to use http://localhost:8888/... in web browser and Postman requests instead of just http://localhost, and you will need to use localhost:8889 when connecting to the database (instead of localhost:3306). |

### *5. Error upon running book.py - cannot import name 'EVENT\_TYPE\_OPENED'*

| *Error:*  ImportError: cannot import name 'EVENT\_TYPE\_OPENED' from 'watchdog.events' |
| --- |
| *Reason:*  Using older version of python (such as 3.9.x) |
| *How to Fix:*  **Permanent Solution:**  Update to latest version of python  **Temporary Solution:**  Remove "debug=True" from app.run()  OR update watchdog  python -m pip install --upgrade watchdog |

1. Path variable is the variable section of the URL  
   e.g. http://localhost:5000/book/**ISBN13**, where the **ISBN13** part is the path variable.  
   Same as when you are setting up the route,  
   e.g. /book/<string:isbn13>, where <string:isb13> is defining the path variable and its type [↑](#footnote-ref-0)
2. See Troubleshooting at the end of the document if you cannot get this working. [↑](#footnote-ref-1)