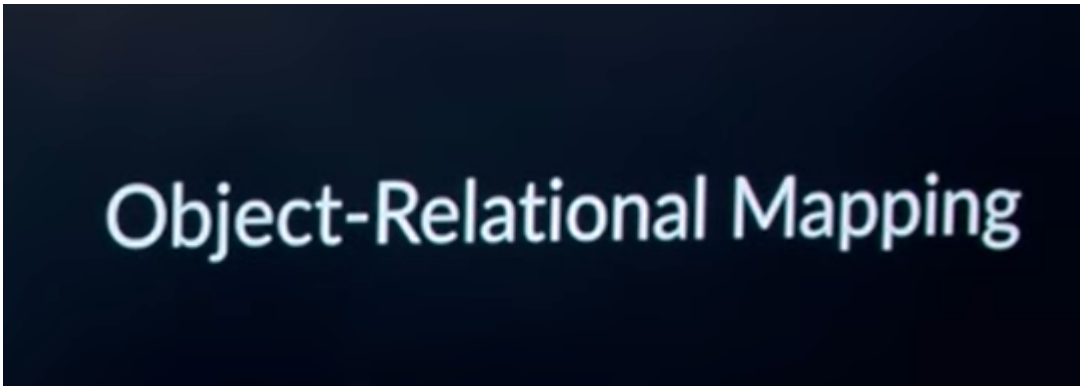
CS50's Web Programming with Python and JavaScript_Lecture 4

April 9, 2023



0.0.1 Create Table

If we have a class like the following, Flask-SQLAlchemy makes it easy to create tables.

```
C: > Users > User > cs50WPPJ_Lecture4 > models.py > ...
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class Flight(db.Model):
    __tablename__ = "flights"
    id = db.Column(db.Integer, primary_key=True)
    origin = db.Column(db.String, nullable=False)
    destination = db.Column(db.String, nullable=False)
    duration = db.Column(db.Integer, nullable=False)


class Passenger(db.Model):
    __tablename__ = "passengers"
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String, nullable=False)
    flight_id = db.Column(db.Integer, db.ForeignKey("flights.id"), nullable=False)

```

db.create_all() makes it easy to create tables without explicitly writing SQL.

```
create.py ×   classes1.py   models.py
C: > Users > User > cs50WPPJ_4 > create.py > ...
import os

from flask import Flask, render_template, request
from models import *

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "postgresql://postgres:12345@localhost:5432/test_database"
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
db.init_app(app) # Tie the database with this flask application

def main():
    db.create_all() # This will create table based on what are inside models.py file

if __name__ == "__main__":
    with app.app_context():
        main()
```

Now, execute the file **create.py**

```
(env) C:\Users\User\cs50WPPJ_4>python create.py
```

Now, we see that the tables **flights** and **passengers** tables are created.

```
postgres=# \c test_database
You are now connected to database "test_database" as user "postgres".
test_database=# \dt
           List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | flights    | table | postgres
 public | passengers | table | postgres
(2 rows)


test_database=#
```

0.0.2 Insert

```
INSERT INTO flights
    (origin, destination, duration)
    VALUES ('New York', 'Paris', 540)
```

```
flight = Flight(origin="New York",
                destination="Paris",
                duration=540)
db.session.add(flight)
```

0.0.3 SELECT

```
SELECT * FROM flights;
```

```
Flight.query.all()
```

```
SELECT * FROM flights
    WHERE origin = 'Paris';
```

```
Flight.query.filter_by(origin="Paris").all()
```

0.0.4 Insert with explicit SQL

import.py | create.py | import0.py × | import1.py

C: > Users > User > cs50WPPJ_4 > import0.py > ...

```python
import csv
import os

from sqlalchemy import create_engine
from sqlalchemy.orm import scoped_session, sessionmaker

engine = create_engine("postgresql://postgres:12345@localhost:5432/test_database")
db = scoped_session(sessionmaker(bind=engine))

def main():
    f = open("flights.csv")
    reader = csv.reader(f)
    for origin, destination, duration in reader:
        db.execute("INSERT INTO flights (origin, destination, duration) VALUES (:origin, :destination, :
                   {"origin": origin, "destination": destination, "duration": duration})
        print(f"Added flight from {origin} to {destination} lasting {duration} minutes.")
    db.commit()

if __name__ == "__main__":
    main()
```

0.0.5 Insert without explicit SQL or with ORM

import.py | create.py | import0.py | import1.py

C: > Users > User > cs50WPPJ_4 > import1.py > ...

```python
import csv
import os

from flask import Flask, render_template, request
from models import *

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "postgresql://postgres:12345@localhost:5432/test_database"
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
db.init_app(app)

def main():
    f = open("flights.csv")
    reader = csv.reader(f)
    for origin, destination, duration in reader:
        flight = Flight(origin=origin, destination=destination, duration=duration)
        db.session.add(flight)
        print(f"Added flight from {origin} to {destination} lasting {duration} minutes.")
    db.session.commit()

if __name__ == "__main__":
    with app.app_context():
        main()
```

```
(env) C:\Users\User\cs50WPPJ_4>python create.py

(env) C:\Users\User\cs50WPPJ_4>python import1.py
Added flight from Paris to New York lasting 540 minutes.
Added flight from Tokyo to Shanghai lasting 185 minutes.
Added flight from Seoul to Mexico City lasting 825 minutes.
Added flight from Mexico City to Lima lasting 350 minutes.
Added flight from Hong Kong to Shanghai lasting 130 minutes.
```

0.0.6 SELECT with explicit SQL

import.py | create.py | import0.py | import1.py | list0.py | list1.py

C: > Users > User > cs50WPPJ_4 > list0.py > ...

```python
import os

from sqlalchemy import create_engine
from sqlalchemy.orm import scoped_session, sessionmaker

engine = create_engine("postgresql://postgres:12345@localhost:5432/test_database")
db = scoped_session(sessionmaker(bind=engine))

def main():
    flights = db.execute("SELECT origin, destination, duration FROM flights").fetchall()
    for flight in flights:
        print(f"{flight.origin} to {flight.destination}, {flight.duration} minutes.")

if __name__ == "__main__":
    main()
```

0.0.7 SELECT without explicit SQL or with ORM

list1.py

C: > Users > User > cs50WPPJ_4 > list1.py > main

```python
import os

from flask import Flask, render_template, request
from models import *

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "postgresql://postgres:12345@localhost:5432/test_database"
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
db.init_app(app)

def main():
    flights = Flight.query.all() # flights is a list of Flight object
    for flight in flights:
        print(f"{flight.origin} to {flight.destination}, {flight.duration} minutes.")

if __name__ == "__main__":
    with app.app_context():
        main()

```

```
(env) C:\Users\User\cs50WPPJ_4>python list1.py
Paris to New York, 540 minutes.
Tokyo to Shanghai, 185 minutes.
Seoul to Mexico City, 825 minutes.
Mexico City to Lima, 350 minutes.
Hong Kong to Shanghai, 130 minutes.
```

0.0.8 LIMIT

```
SELECT * FROM flights
    WHERE origin = 'Paris' LIMIT 1;
```

```
Flight.query.filter_by(origin="Paris").first()
```

0.0.9 .count – Number of rows

```
SELECT COUNT(*) FROM flights
    WHERE origin = 'Paris';
```

```
Flight.query.filter_by(origin="Paris").count()
```

0.0.10 SELECT WHERE

```
SELECT * FROM flights WHERE id = 28;
```

```
Flight.query.filter_by(id=28).first()
```

0.0.11 Or alternatively, easier way

```
Flight.query.get(28)
```

0.1 Updating a table

```
UPDATE flights SET duration = 280
    WHERE id = 6;
```

```
flight = Flight.query.get(6)
flight.duration = 280
```

0.1.1 DELETE

```
DELETE FROM flights WHERE id = 28;
```

```
flight = Flight.query.get(28)
db.session.delete(flight)
```

0.1.2 COMMIT

```
COMMIT;
```

```
db.session.commit()
```

0.1.3 ORDER BY

```
SELECT * FROM flights
    ORDER BY origin;
```

```
Flight.query.order_by(Flight.origin).all()
```

DESC ORDER

```
SELECT * FROM flights
    ORDER BY origin DESC;
```

```
Flight.query.order_by(Flight.origin.desc()).all()
```

0.1.4 ! – Not equal to

filter (not filter_by)

```
SELECT * FROM flights
    WHERE origin != "Paris"
```

```
Flight.query.filter(
    Flight.origin != "Paris").all()
```

0.1.5 LIKE

```
SELECT * FROM flights
    WHERE origin LIKE "%a%"
```

```
Flight.query.filter(
    Flight.origin.like("%a%")).all()
```

0.1.6 IN vs. in_

```
SELECT * FROM flights
    WHERE origin IN ('Tokyo', 'Paris');
```

```
Flight.query.filter(
    Flight.origin.in_(
        ["Tokyo", "Paris"])).all()
```

0.1.7 And vs. and_

```
SELECT * FROM flights
    WHERE origin = "Paris"
    AND duration > 500;
```

```
Flight.query.filter(
    and_(Flight.origin == "Paris",
         Flight.duration > 500)).all()
```

0.1.8 OR vs. or_

from sqlalchemy import or_

```
SELECT * FROM flights
    WHERE origin = "Paris"
    OR duration > 500;
```

```
Flight.query.filter(
    or_(Flight.origin == "Paris",
        Flight.duration > 500)).all()
```

0.1.9 JOIN

```
SELECT * FROM flights JOIN passengers
    ON flights.id = passengers.flight_id;
```

```
db.session.query(Flight, Passenger).filter(
  Flight.id == Passenger.flight_id).all()
```

0.1.10 Version 1

> Users > User > cs50WPPJ_4 > airline2 > models.py > ...

```python
import os

from flask import Flask
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()


class Flight(db.Model):
    __tablename__ = "flights"
    id = db.Column(db.Integer, primary_key=True)
    origin = db.Column(db.String, nullable=False)
    destination = db.Column(db.String, nullable=False)
    duration = db.Column(db.Integer, nullable=False)


class Passenger(db.Model):
    __tablename__ = "passengers"
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String, nullable=False)
    flight_id = db.Column(db.Integer, db.ForeignKey("flights.id"), nullable=False)

```

model.py

```python
@app.route("/book", methods=["POST"])
def book():
    """Book a flight."""

    # Get form information.
    name = request.form.get("name")
    try:
        flight_id = int(request.form.get("flight_id"))
    except ValueError:
        return render_template("error.html", message="Invalid flight number.")

    # Make sure flight exists.
    if db.execute("SELECT * FROM flights WHERE id = :id", {"id": flight_id}).rowcount == 0:
        return render_template("error.html", message="No such flight with that id.")
    db.execute("INSERT INTO passengers (name, flight_id) VALUES (:name, :flight_id)",
            {"name": name, "flight_id": flight_id})
    db.commit()
    return render_template("success.html")
```

0.1.11 Version 2

```python
@app.route("/book", methods=["POST"])
def book():
    """Book a flight."""

    # Get form information.
    name = request.form.get("name")
    try:
        flight_id = int(request.form.get("flight_id"))
    except ValueError:
        return render_template("error.html", message="Invalid flight number.")

    # Make sure the flight exists.
    flight = Flight.query.get(flight_id)
    if flight is None:
        return render_template("error.html", message="No such flight with that id.")

    # Add passenger.
    passenger = Passenger(name=name, flight_id=flight_id)
    db.session.add(passenger)
    db.session.commit()
    return render_template("success.html")
```

0.1.12 Version 3 – Adding passenger facility as a function in the Flight class in the models.py

application.py C:\...\airline2 | application.py C:\...\airline3 | models.py C:\...\airline3 × | models.py C:\...\airline2

: > Users > User > cs50WPPJ_4 > airline3 > models.py > ...

```python
import os

from flask import Flask
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()


class Flight(db.Model):
    __tablename__ = "flights"
    id = db.Column(db.Integer, primary_key=True)
    origin = db.Column(db.String, nullable=False)
    destination = db.Column(db.String, nullable=False)
    duration = db.Column(db.Integer, nullable=False)

    def add_passenger(self, name):
        p = Passenger(name=name, flight_id=self.id)
        db.session.add(p)
        db.session.commit()


class Passenger(db.Model):
    __tablename__ = "passengers"
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String, nullable=False)
    flight_id = db.Column(db.Integer, db.ForeignKey("flights.id"), nullable=False)

```

```python
@app.route("/book", methods=["POST"])
def book():
    """Book a flight."""

    # Get form information.
    name = request.form.get("name")
    try:
        flight_id = int(request.form.get("flight_id"))
    except ValueError:
        return render_template("error.html", message="Invalid flight number.")

    # Make sure the flight exists.
    flight = Flight.query.get(flight_id)
    if not flight:
        return render_template("error.html", message="No such flight with that id.")

    # Add passenger.
    flight.add_passenger(name)
    return render_template("success.html")
```

0.1.13 Without relation or with previous models.py

```python
@app.route("/flights/<int:flight_id>")
def flight(flight_id):
    """List details about a single flight."""

    # Make sure flight exists.
    flight = Flight.query.get(flight_id)
    if flight is None:
        return render_template("error.html", message="No such flight.")

    # Get all passengers.
    passengers = Passenger.query.filter_by(flight_id=flight_id).all()
    return render_template("flight.html", flight=flight, passengers=passengers)
```

0.1.14 Modified table definition

```
application.py C:\...\airline2   application.py C:\...\airline3   models.py C:\...\airline4 ×   models.py C:\...\airline3
C: > Users > User > cs50WPPJ_4 > airline4 > models.py > ...
import os

from flask import Flask
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()


class Flight(db.Model):
    __tablename__ = "flights"
    id = db.Column(db.Integer, primary_key=True)
    origin = db.Column(db.String, nullable=False)
    destination = db.Column(db.String, nullable=False)
    duration = db.Column(db.Integer, nullable=False)
    passengers = db.relationship("Passenger", backref="flight", lazy=True)

    def add_passenger(self, name):
        p = Passenger(name=name, flight_id=self.id)
        db.session.add(p)
        db.session.commit()


class Passenger(db.Model):
    __tablename__ = "passengers"
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String, nullable=False)
    flight_id = db.Column(db.Integer, db.ForeignKey("flights.id"), nullable=False)

```

```
@app.route("/flights/<int:flight_id>")
def flight(flight_id):
    """List details about a single flight."""

    # Make sure flight exists.
    flight = Flight.query.get(flight_id)
    if flight is None:
        return render_template("error.html", message="No such flight.")

    # Get all passengers.
    passengers = flight.passengers
    return render_template("flight.html", flight=flight, passengers=passengers)
```

0.1.15 Relationship using new modified table definition with relation

```
SELECT * FROM passengers
    WHERE flight_id = 1;
```

```
Flight.query.get(1).passengers
```

```
SELECT * FROM flights JOIN passengers
    ON flights.id = passengers.flight_id
    WHERE passengers.name = 'Alice';
```

```
Passenger.query.filter_by(name="Alice") \
                .first().flight
```

0.1.16 APIs

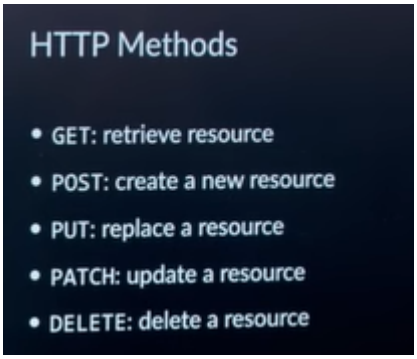
- Protocols for communicating between either different web applications or different parts of the see web application or different components of different applications
- Perform actions in other spaces
- Allows different components of a web application to communicate with each other

0.1.17 JSON – pass data from one application to another

```
{
    "origin": "Tokyo",
    "destination": "Shanghai",
    "duration": 185
}
```

```
{
    "origin": "Tokyo",
    "destination": "Shanghai",
    "duration": 185,
    "passengers": ["Alice", "Bob"]
}
```

```
{
    "origin": {
        "city": "Tokyo",
        "code": "HND"
    },
    "destination", {
        "city": "Shanghai",
        "code": "PVG"
    },
    "duration": 185,
    "passengers": ["Alice", "Bob"]
}
```



0.1.18 requests

requests library makes iut easy for us to make all of these HTTP requests to different web servers.

```
pip install requests
```

To use this, use **import requets** at the top of **.py** file.

```
import requests
```

```
requests.get(url)

requests.post(url)

requests.put(url)

requests.patch(url)

requests.delete(url)
```

We will work with API from this site for foreign exchange rate now: https://fixer.io/

My API key for this site is:

2f6a6520d5ff09563e0a6a379c07242c

http://data.fixer.io/api/latest?access_key=2f6a6520d5ff09563e0a6a379c07242c

I am also using **openweathermap** https://openweathermap.org/api and its API key for me is:

6e6504bcf4521fe73da79851dd59ad95

To get data, type the API key at the end of the call

https://api.openweathermap.org/data/2.5/onecall?lat=33.441792&lon=-94.037689&exclude=hourly,daily&appid=6e6504bcf4521fe73da79851dd59ad95


```
{"lat":33.44,"lon":-94.04,"timezone":"America/Chicago","timezone_offset":-18000,"current":
{"dt":1590147634,"sunrise":1590145882,"sunset":1590196481,"temp":294.49,"feels_like":296.18,"pressure":1011,"humidity":94,"dew_point":293.48,"u
vi":10.93,"clouds":75,"visibility":14484,"wind_speed":3.1,"wind_deg":170,"weather":[{"id":803,"main":"Clouds","description":"broken
clouds","icon":"04d"}]},"minutely":[{"dt":1590147660,"precipitation":0},{"dt":1590147720,"precipitation":0},
{"dt":1590147780,"precipitation":0},{"dt":1590147840,"precipitation":0},{"dt":1590147900,"precipitation":0},
{"dt":1590147960,"precipitation":0},{"dt":1590148020,"precipitation":0},{"dt":1590148080,"precipitation":0},
{"dt":1590148140,"precipitation":0},{"dt":1590148200,"precipitation":0},{"dt":1590148260,"precipitation":0},
{"dt":1590148320,"precipitation":0},{"dt":1590148380,"precipitation":0},{"dt":1590148440,"precipitation":0},
{"dt":1590148500,"precipitation":0},{"dt":1590148560,"precipitation":0},{"dt":1590148620,"precipitation":0},
{"dt":1590148680,"precipitation":0},{"dt":1590148740,"precipitation":0},{"dt":1590148800,"precipitation":0},
{"dt":1590148860,"precipitation":0},{"dt":1590148920,"precipitation":0},{"dt":1590148980,"precipitation":0},
{"dt":1590149040,"precipitation":0},{"dt":1590149100,"precipitation":0},{"dt":1590149160,"precipitation":0},
{"dt":1590149220,"precipitation":0},{"dt":1590149280,"precipitation":0},{"dt":1590149340,"precipitation":0},
{"dt":1590149400,"precipitation":0},{"dt":1590149460,"precipitation":0},{"dt":1590149520,"precipitation":0},
{"dt":1590149580,"precipitation":0},{"dt":1590149640,"precipitation":0},{"dt":1590149700,"precipitation":0},
{"dt":1590149760,"precipitation":0},{"dt":1590149820,"precipitation":0},{"dt":1590149880,"precipitation":0},
{"dt":1590149940,"precipitation":0},{"dt":1590150000,"precipitation":0},{"dt":1590150060,"precipitation":0},
{"dt":1590150120,"precipitation":0},{"dt":1590150180,"precipitation":0},{"dt":1590150240,"precipitation":0},
{"dt":1590150300,"precipitation":0},{"dt":1590150360,"precipitation":0},{"dt":1590150420,"precipitation":0},
{"dt":1590150480,"precipitation":0},{"dt":1590150540,"precipitation":0},{"dt":1590150600,"precipitation":0},
{"dt":1590150660,"precipitation":0},{"dt":1590150720,"precipitation":0},{"dt":1590150780,"precipitation":0},
{"dt":1590150840,"precipitation":0},{"dt":1590150900,"precipitation":0},{"dt":1590150960,"precipitation":0},
{"dt":1590151020,"precipitation":0},{"dt":1590151080,"precipitation":0},{"dt":1590151140,"precipitation":0},
{"dt":1590151200,"precipitation":0},{"dt":1590151260,"precipitation":0}]}
```

ation.py C:\...\airline3 | application.py C:\...\airline4 | models.py C:\...\airline4 | google.py

C: > Users > User > cs50WPPJ_4 > currency0.py > ...

```python
import requests

def main():
    res = requests.get("https://api.fixer.io/latest?base=USD&symbols=EUR")
    if res.status_code != 200:
        raise Exception("ERROR: API request unsuccessful.")
    data = res.json()
    print(data)

if __name__ == "__main__":
    main()
```

```
src $ python currency0.py
{'base': 'USD', 'date': '2018-02-26', 'rates': {'EUR': 0.81169}}
src $
```

Status Codes

- 200 OK
- 201 Created
- 400 Bad Request
- 403 Forbidden
- 404 Not Found
- 405 Method Not Allowed
- 422 Unprocessable Entity
- ...

application.py C:\...\airline4 | models.py C:\...\airline4 | google.py | currency0.py | currency1.py

C: > Users > User > cs50WPPJ_4 > currency1.py > ...

```python
import requests

def main():
    res = requests.get("https://api.fixer.io/latest?base=USD&symbols=EUR")
    if res.status_code != 200:
        raise Exception("ERROR: API request unsuccessful.")
    data = res.json()
    rate = data["rates"]["EUR"]
    print(f"1 USD is equal to {rate} EUR")

if __name__ == "__main__":
    main()
```

1. bash

```
src $ python currency1.py
1 USD is equal to 0.81169 EUR
src $
```

...\airline4 | models.py C:\...\airline4 | google.py | currency0.py | currency1.py | currency2.py ×

C: > Users > User > cs50WPPJ_4 > currency2.py > ...

```python
import requests

def main():
    base = input("First Currency: ")
    other = input("Second Currency: ")
    res = requests.get("https://api.fixer.io/latest",
                       params={"base": base, "symbols": other})
    if res.status_code != 200:
        raise Exception("ERROR: API request unsuccessful.")
    data = res.json()
    rate = data["rates"][other]
    print(f"1 {base} is equal to {rate} {other}")

if __name__ == "__main__":
    main()
```

iTerm2 Shell Edit View Session Profiles Toolbelt Window Help

```
src $ python currency2.py
First Currency: USD
Second Currency: JPY
1 USD is equal to 106.82 JPY
src $
```

0.1.19 Creating our own API

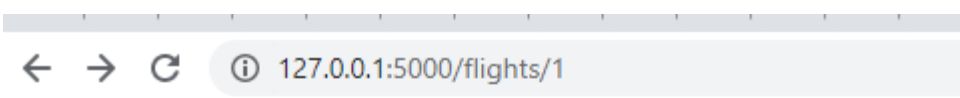
Modified airline To make json, use **jsonify** and at the top of the **.py** file, import **jsonify**

from Flask import jsonify

```python
@app.route("/api/flights/<int:flight_id>")
def flight_api(flight_id):
    """Return details about a single flight."""

    # Make sure flight exists.
    flight = Flight.query.get(flight_id)
    if flight is None:
        return jsonify({"error": "Invalid flight_id"}), 422

    # Get all passengers.
    passengers = flight.passengers
    names = []
    for passenger in passengers:
        names.append(passenger.name)
    return jsonify({
            "origin": flight.origin,
            "destination": flight.destination,
            "duration": flight.duration,
            "passengers": names
        })
```



Flight Details

- Origin: Paris
- Destination: New York
- Duration: 540 minutes

Passengers

- Mridha Ali
- Shammunul Islam
- Mazid Miah
- Rahmat Ali

127.0.0.1:5000/api/flights/1


```
{
  "destination": "New York",
  "duration": 540,
  "origin": "Paris",
  "passengers": [
    "Mridha Ali",
    "Shammunul Islam",
    "Mazid Miah",
    "Rahmat Ali"
  ]
}
```


Now, from Python, we can use this API.

```
C:\Users\User\cs50WPPJ_4>python
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import requests
>>> res = requests.get("http://127.0.0.1:5000/api/flights/1")
>>> res.json()
{'destination': 'New York', 'duration': 540, 'origin': 'Paris', 'passengers': ['Mridha Ali', 'Shammunul Islam', 'Mazid M
iah', 'Rahmat Ali']}
```

```
>>> data = res.json()
>>> data["passengers"]
['Mridha Ali', 'Shammunul Islam', 'Mazid Miah', 'Rahmat Ali']
>>>
```

```
>>> res = requests.get("http://127.0.0.1:5000/api/flights/100")
>>> res.json()
{'error': 'Invalid flight_id'}
>>>
```

```
>>> res.status_code
422
>>>
```

0.1.20 API Keys

API access restrict access by using API key.

```
[ ]:
```