HIRING

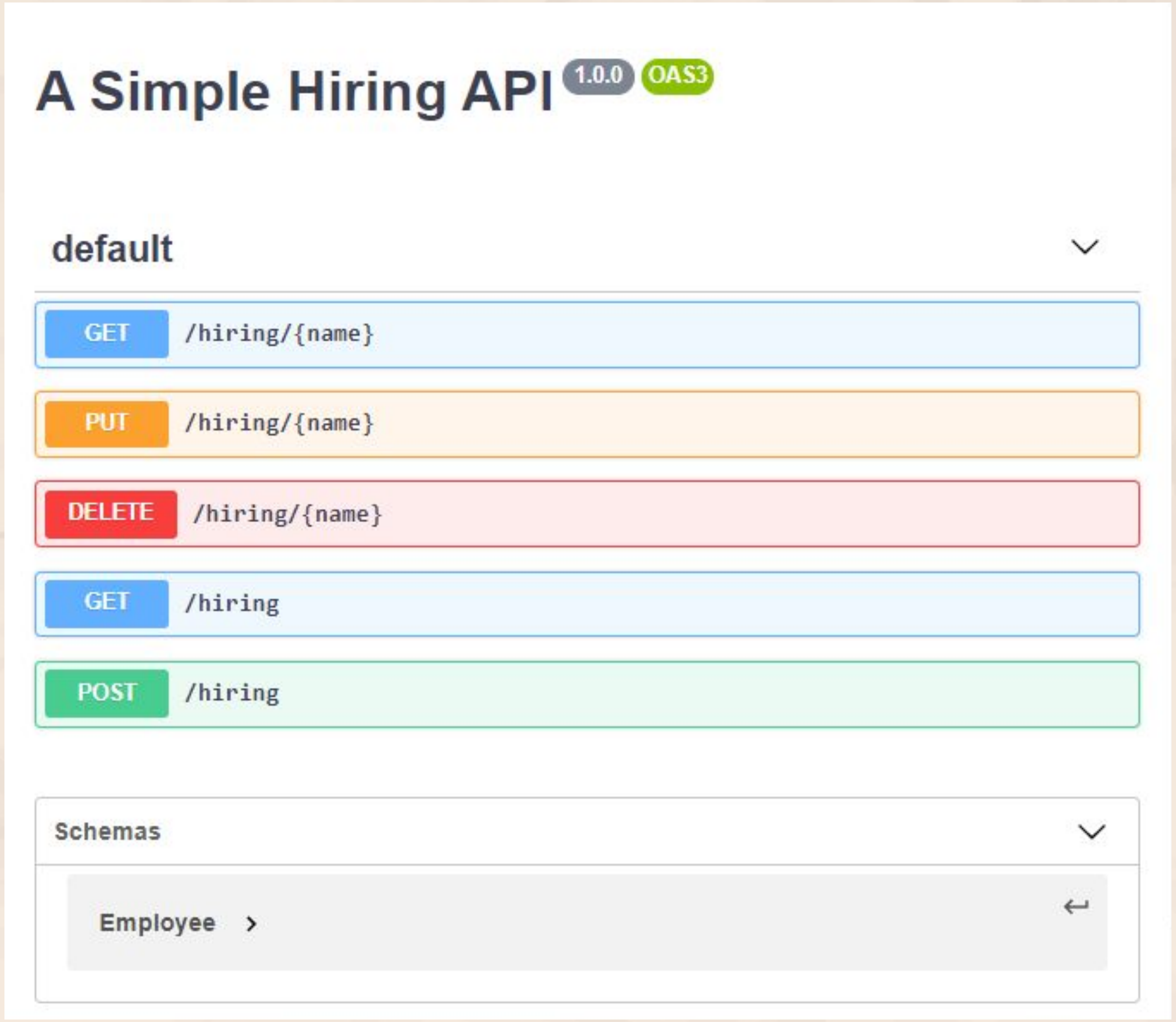
5/2021

19 May 2021

KANNICHA KHAMJRING 61102010135
PAWARIT SRIPIBOON 61102010151
PACHARASIRI SIRIYOM 61102010154

TABLE OF CONTENT

SWAGGER UI



A Simple Hiring API 1.0.0 OAS3

default

GET /hiring/{name}

PUT /hiring/{name}

DELETE /hiring/{name}

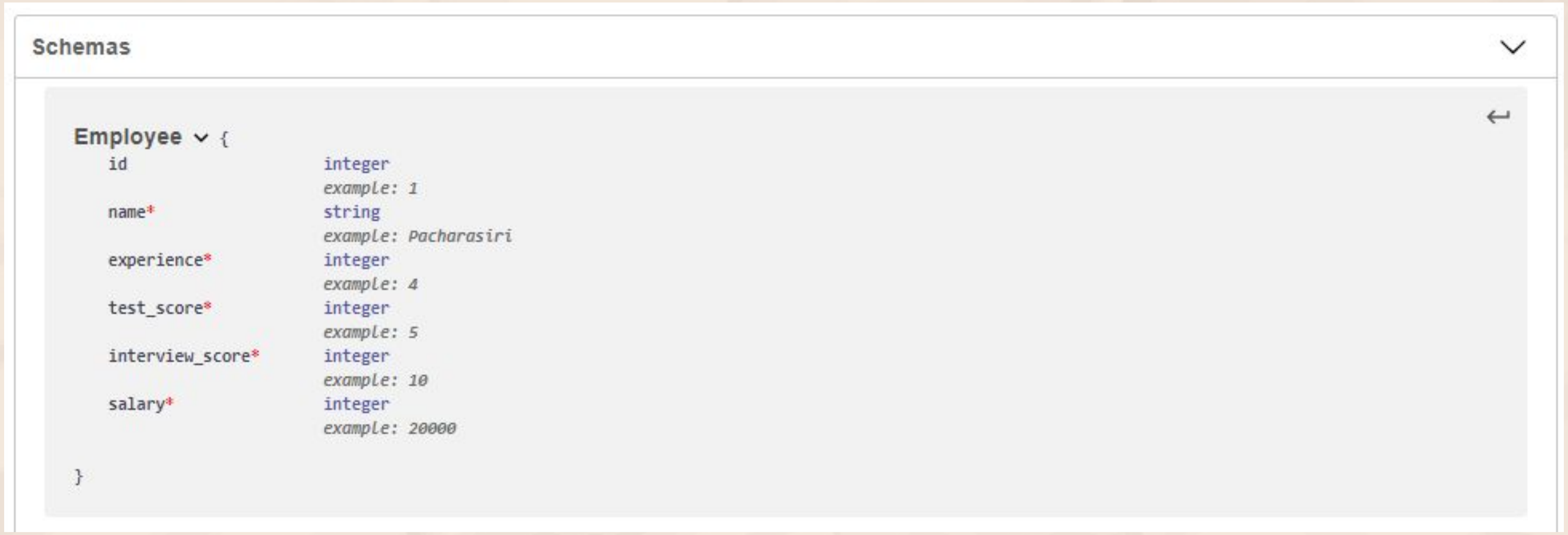
GET /hiring

POST /hiring

Schemas

Employee

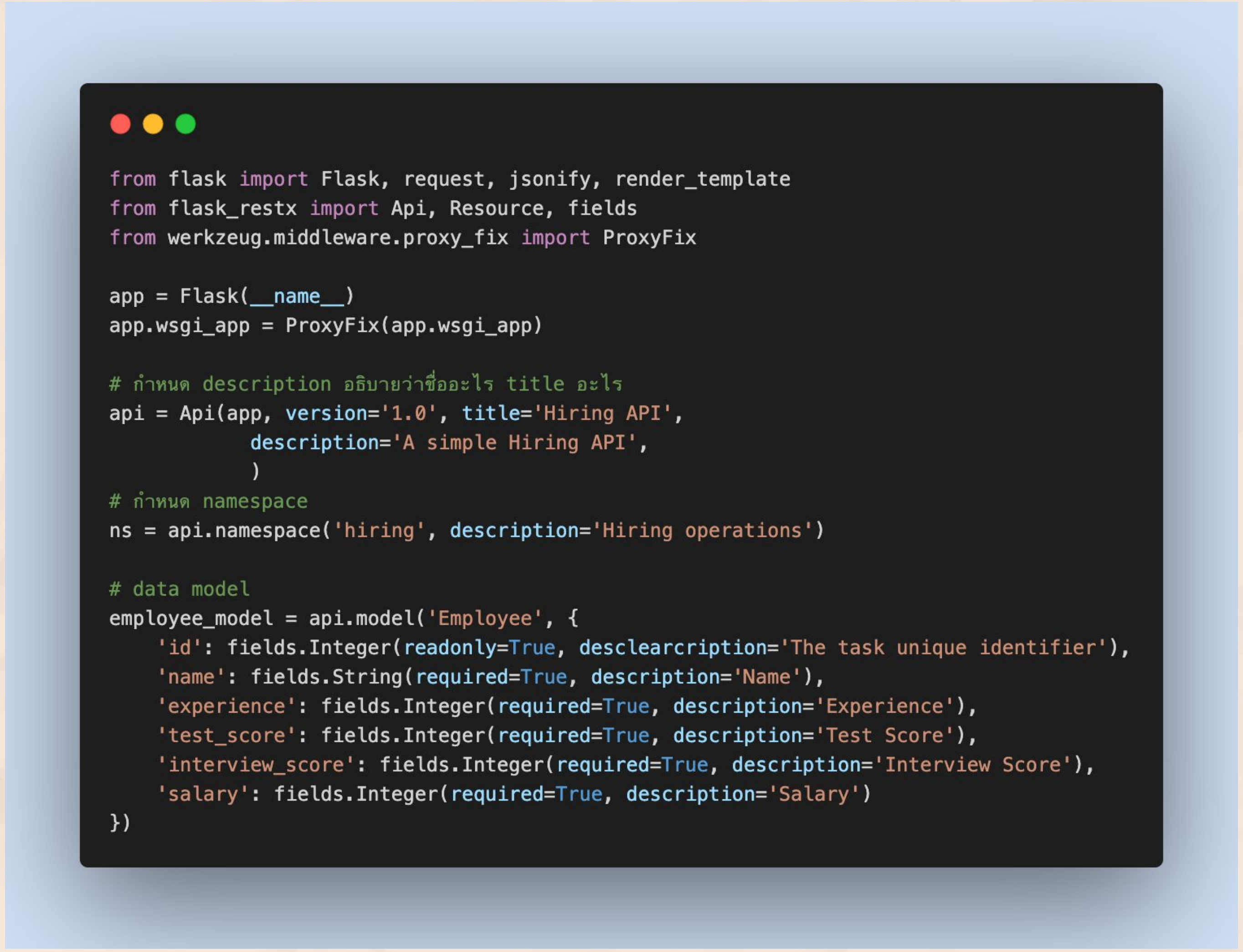
SWAGGER UI



Schemas

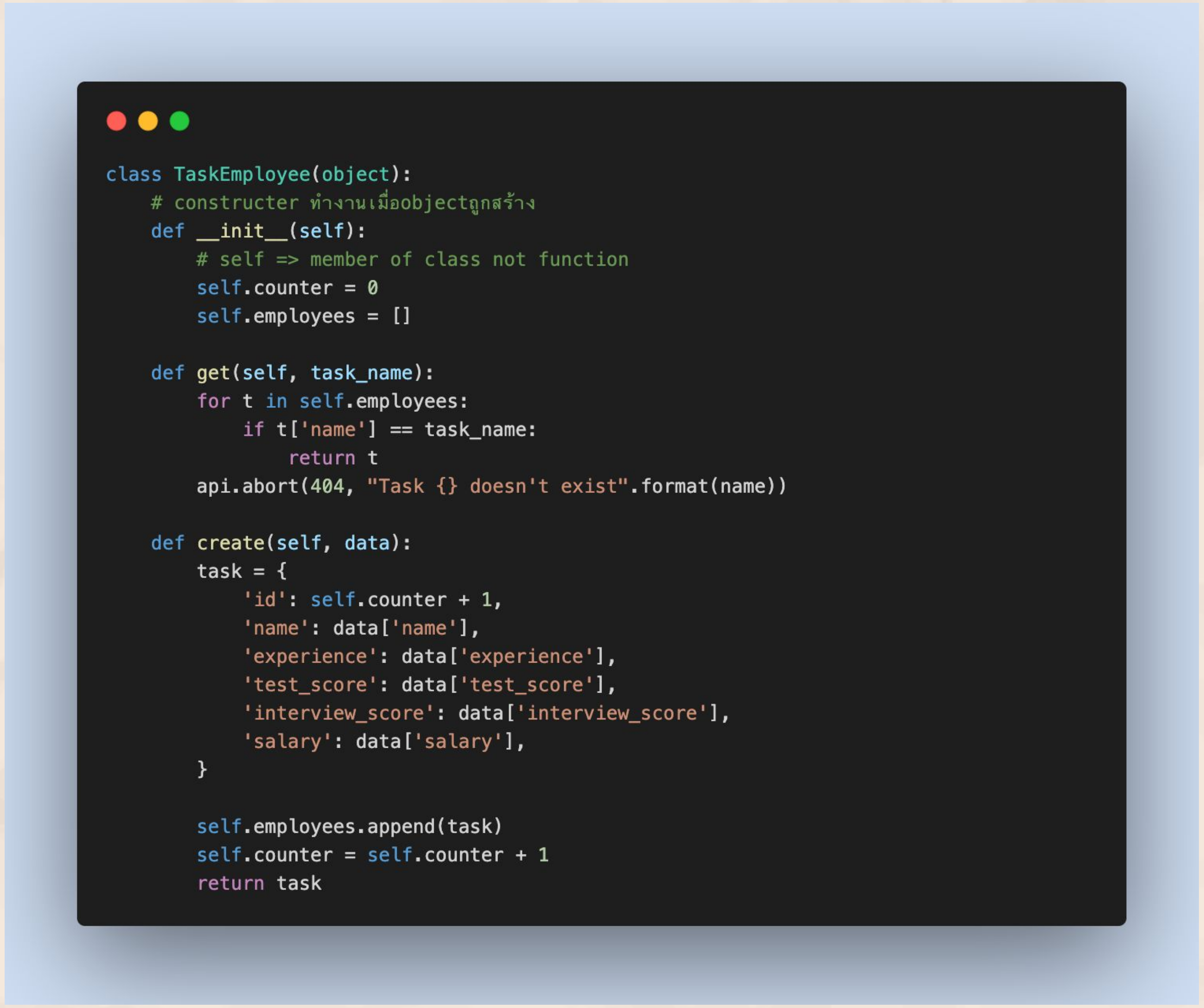
```
Employee {
    id                  integer
                        example: 1
    name*               string
                        example: Pacharasiri
    experience*         integer
                        example: 4
    test_score*         integer
                        example: 5
    interview_score*    integer
                        example: 10
    salary*             integer
                        example: 20000

}
```

Hiring API



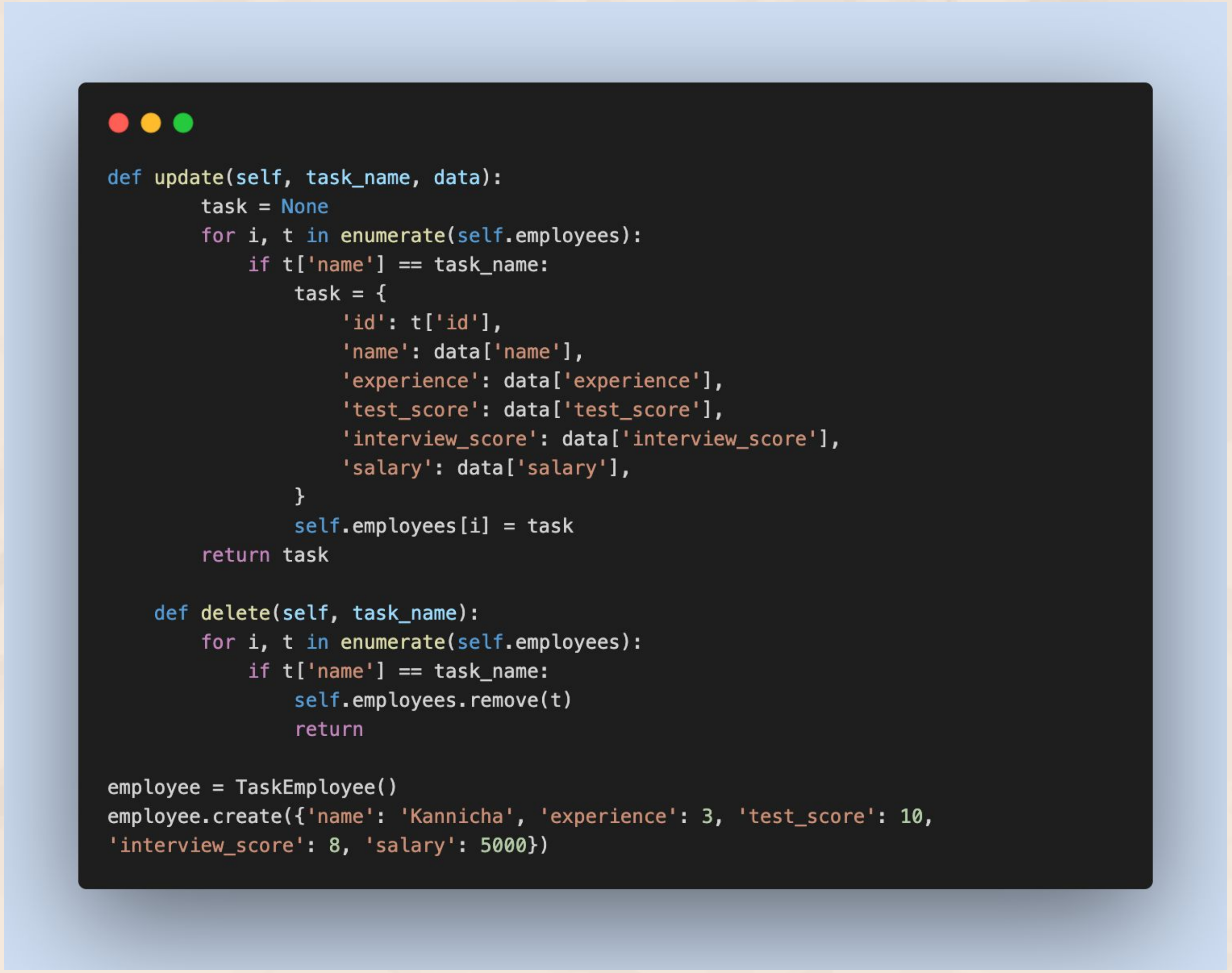
```
from flask import Flask, request, jsonify, render_template
from flask_restx import Api, Resource, fields
from werkzeug.middleware.proxy_fix import ProxyFix

app = Flask(__name__)
app.wsgi_app = ProxyFix(app.wsgi_app)

# กำหนด description อธิบายว่าชื่ออะไร title อะไร
api = Api(app, version='1.0', title='Hiring API',
            description='A simple Hiring API',
            )
# กำหนด namespace
ns = api.namespace('hiring', description='Hiring operations')

# data model
employee_model = api.model('Employee', {
    'id': fields.Integer(readonly=True, desclearcription='The task unique identifier'),
    'name': fields.String(required=True, description='Name'),
    'experience': fields.Integer(required=True, description='Experience'),
    'test_score': fields.Integer(required=True, description='Test Score'),
    'interview_score': fields.Integer(required=True, description='Interview Score'),
    'salary': fields.Integer(required=True, description='Salary')
})
```

Hiring API



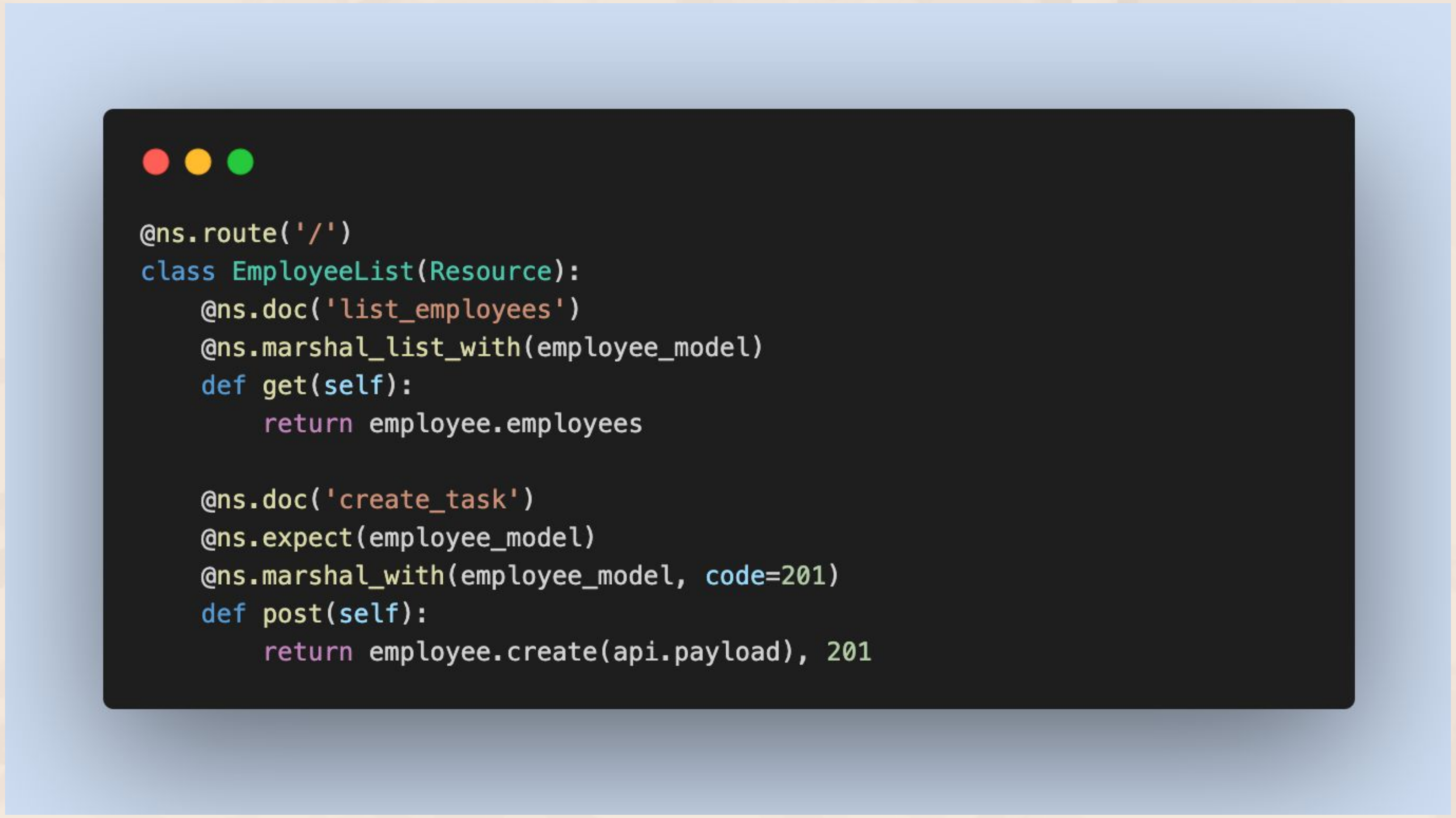
```python
class TaskEmployee(object):
    # constructer ทำงานเมื่อobjectถูกสร้าง
    def __init__(self):
        # self => member of class not function
        self.counter = 0
        self.employees = []

    def get(self, task_name):
        for t in self.employees:
            if t['name'] == task_name:
                return t
        api.abort(404, "Task {} doesn't exist".format(name))

    def create(self, data):
        task = {
            'id': self.counter + 1,
            'name': data['name'],
            'experience': data['experience'],
            'test_score': data['test_score'],
            'interview_score': data['interview_score'],
            'salary': data['salary'],
        }

        self.employees.append(task)
        self.counter = self.counter + 1
        return task
```

Hiring API



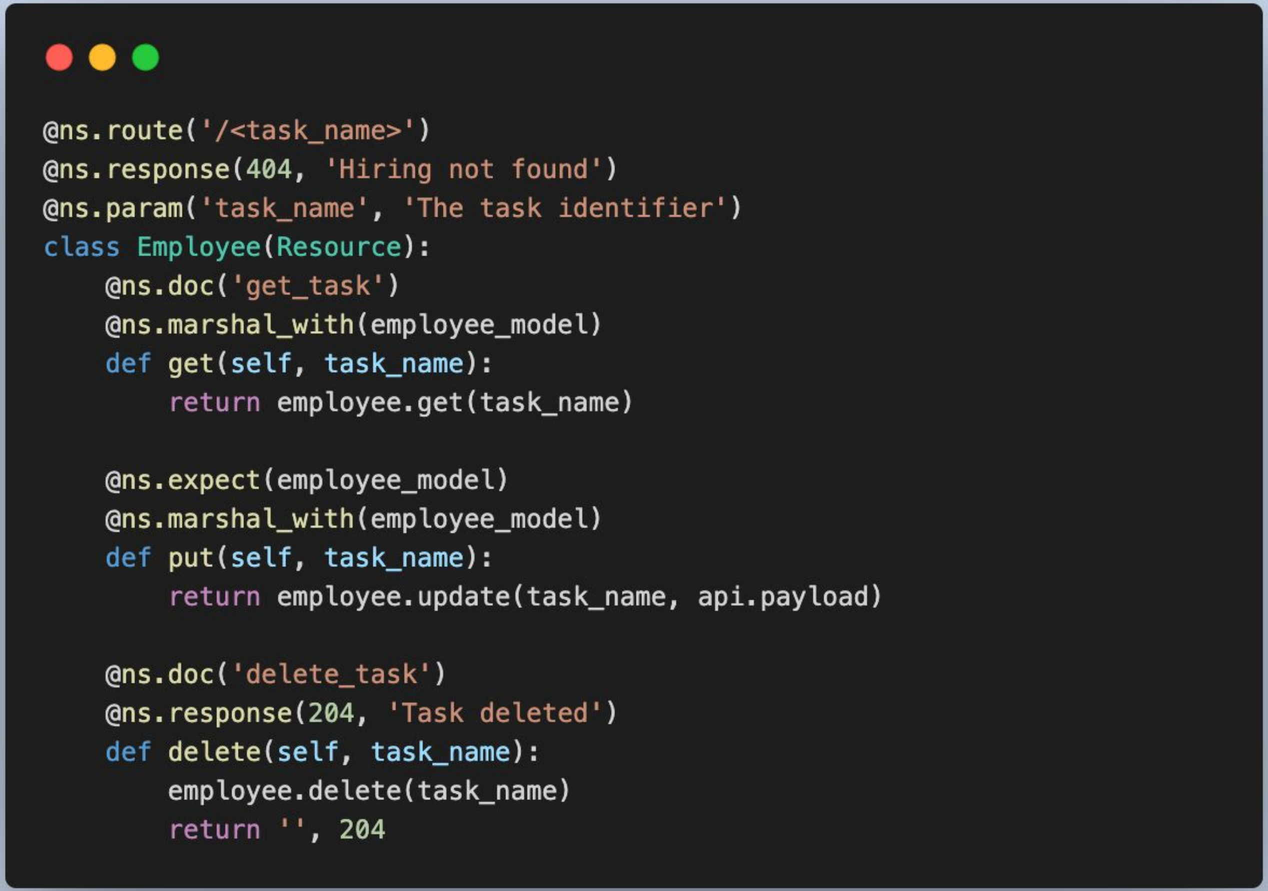
```python
def update(self, task_name, data):
        task = None
        for i, t in enumerate(self.employees):
            if t['name'] == task_name:
                task = {
                    'id': t['id'],
                    'name': data['name'],
                    'experience': data['experience'],
                    'test_score': data['test_score'],
                    'interview_score': data['interview_score'],
                    'salary': data['salary'],
                }
                self.employees[i] = task
        return task

    def delete(self, task_name):
        for i, t in enumerate(self.employees):
            if t['name'] == task_name:
                self.employees.remove(t)
                return

employee = TaskEmployee()
employee.create({'name': 'Kannicha', 'experience': 3, 'test_score': 10,
'interview_score': 8, 'salary': 5000})
```

Hiring API



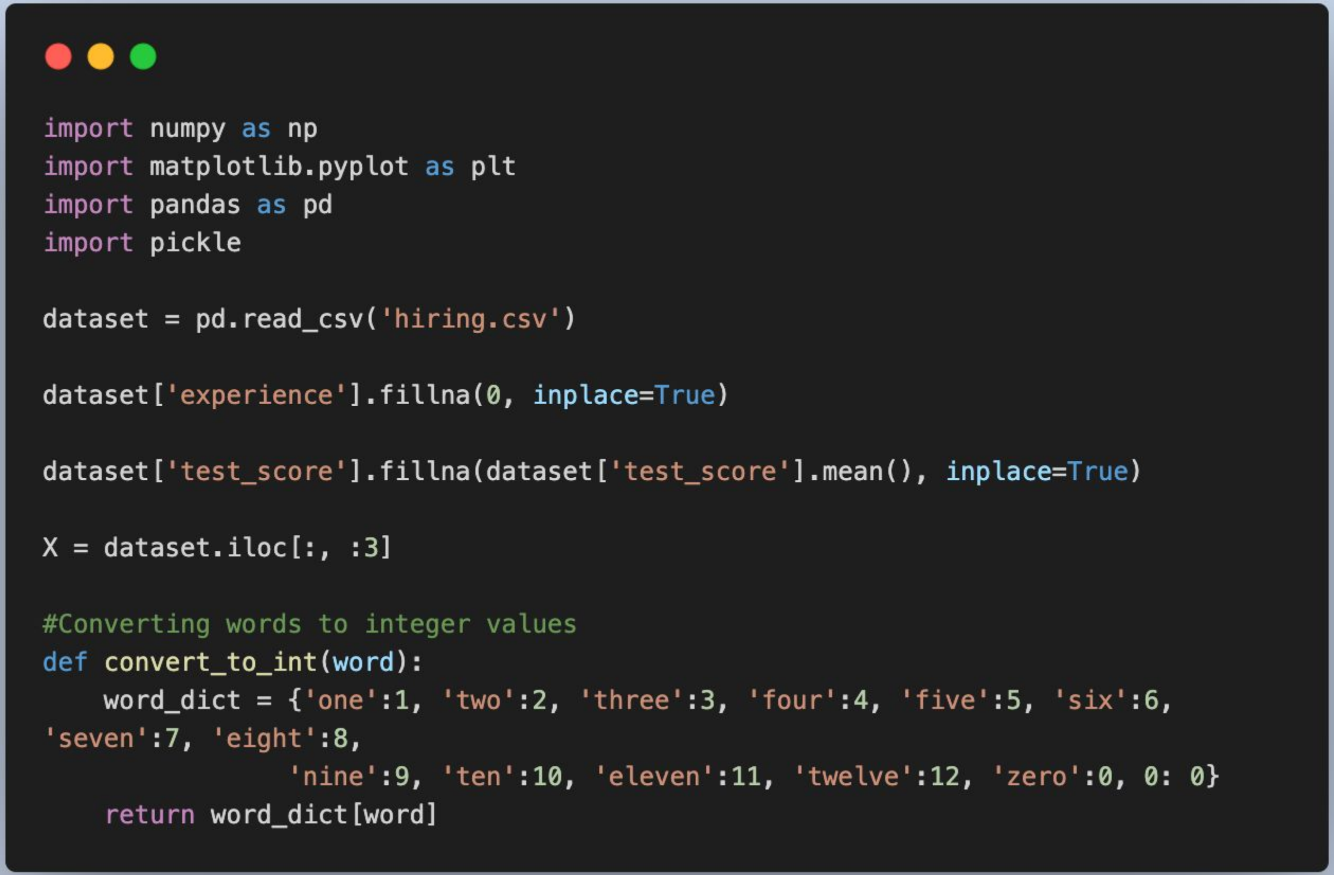
```python
@ns.route('/')
class EmployeeList(Resource):
    @ns.doc('list_employees')
    @ns.marshal_list_with(employee_model)
    def get(self):
        return employee.employees

    @ns.doc('create_task')
    @ns.expect(employee_model)
    @ns.marshal_with(employee_model, code=201)
    def post(self):
        return employee.create(api.payload), 201
```

Hiring API



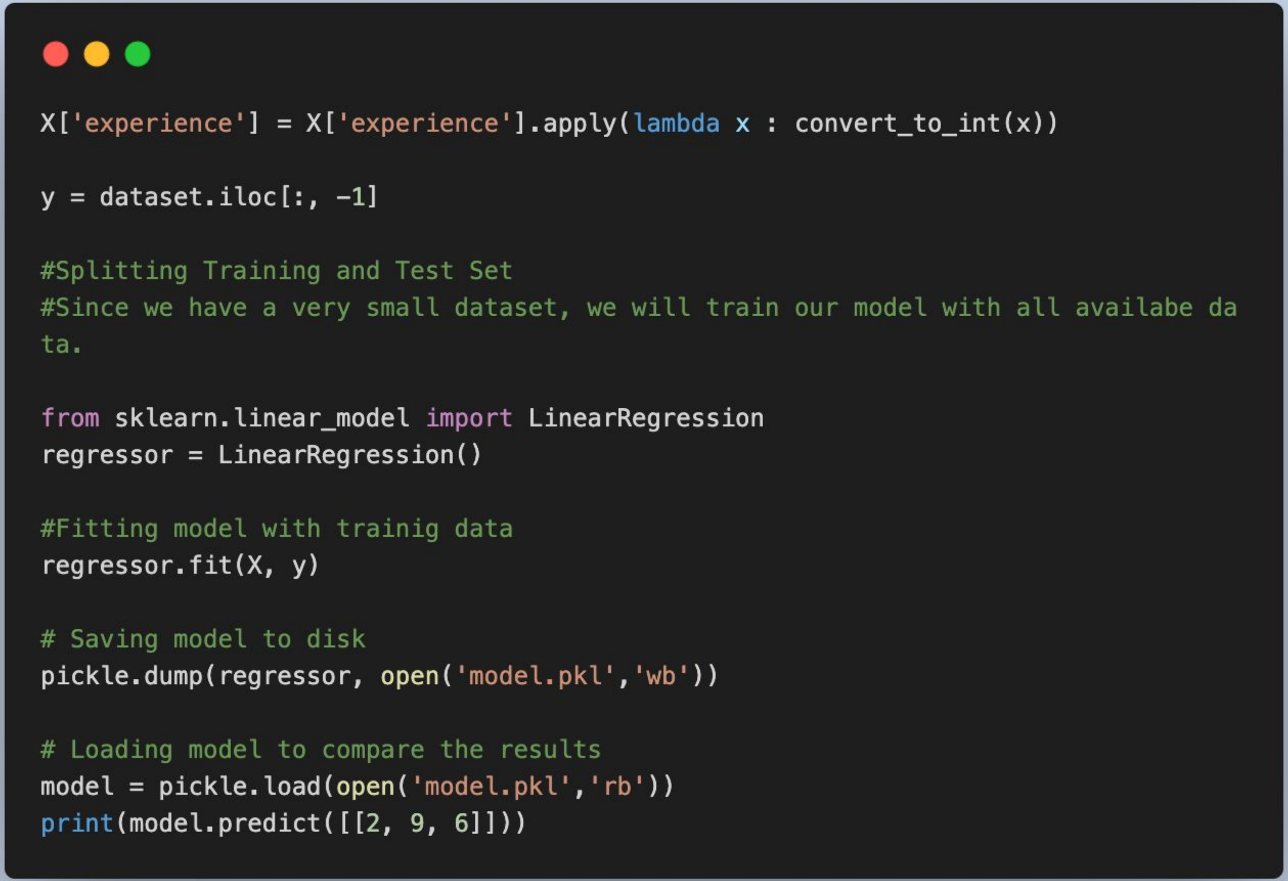
```python
@ns.route('/<task_name>')
@ns.response(404, 'Hiring not found')
@ns.param('task_name', 'The task identifier')
class Employee(Resource):
    @ns.doc('get_task')
    @ns.marshal_with(employee_model)
    def get(self, task_name):
        return employee.get(task_name)

    @ns.expect(employee_model)
    @ns.marshal_with(employee_model)
    def put(self, task_name):
        return employee.update(task_name, api.payload)

    @ns.doc('delete_task')
    @ns.response(204, 'Task deleted')
    def delete(self, task_name):
        employee.delete(task_name)
        return '', 204
```

MACHINE LEARNING



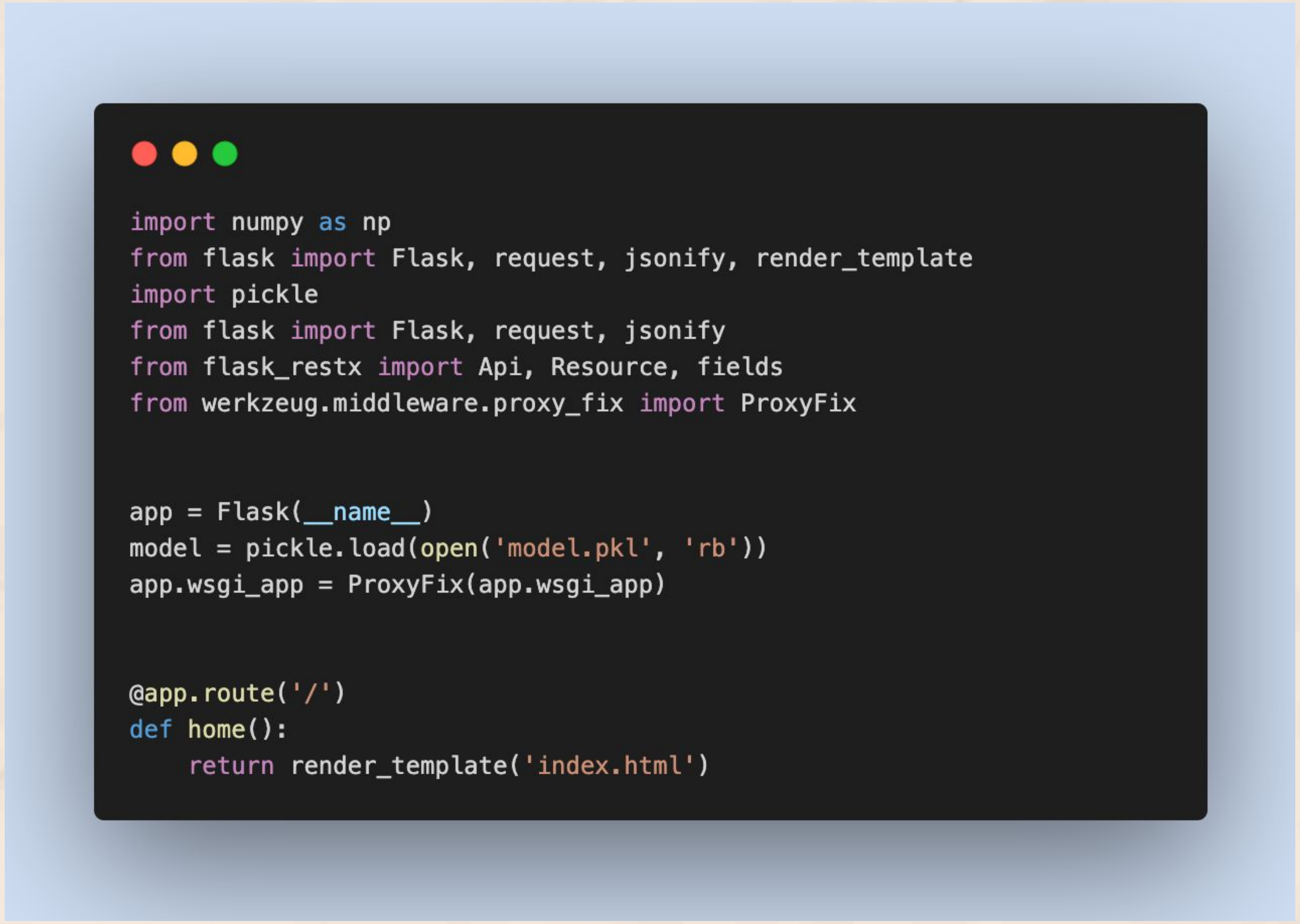
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pickle

dataset = pd.read_csv('hiring.csv')

dataset['experience'].fillna(0, inplace=True)

dataset['test_score'].fillna(dataset['test_score'].mean(), inplace=True)

X = dataset.iloc[:, :3]

#Converting words to integer values
def convert_to_int(word):
    word_dict = {'one':1, 'two':2, 'three':3, 'four':4, 'five':5, 'six':6,
'seven':7, 'eight':8,
                'nine':9, 'ten':10, 'eleven':11, 'twelve':12, 'zero':0, 0: 0}
    return word_dict[word]
```

MACHINE LEARNING



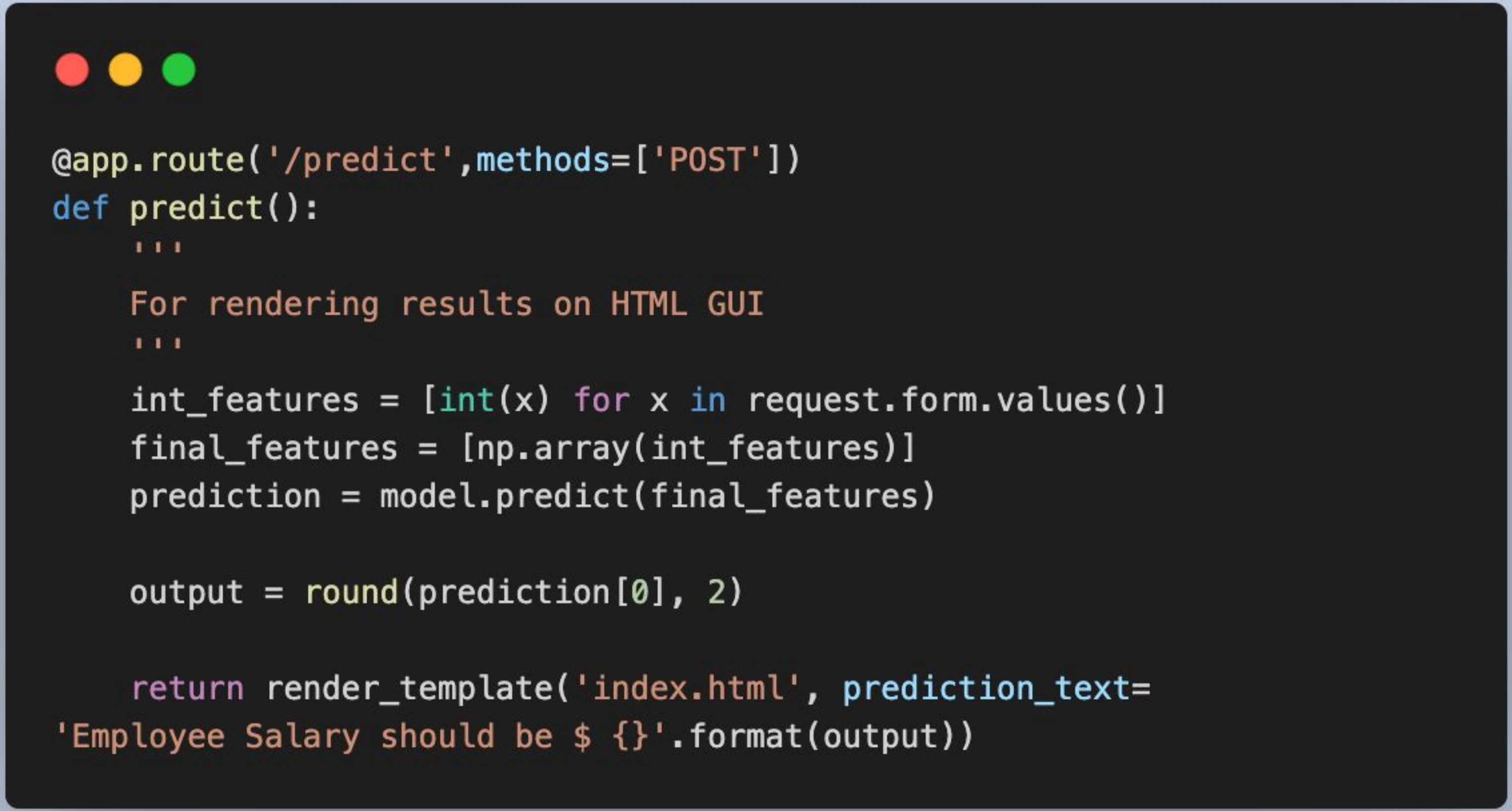
```
X['experience'] = X['experience'].apply(lambda x : convert_to_int(x))

y = dataset.iloc[:, -1]

#Splitting Training and Test Set
#Since we have a very small dataset, we will train our model with all availabe da
ta.

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()

#Fitting model with trainig data
regressor.fit(X, y)

# Saving model to disk
pickle.dump(regressor, open('model.pkl','wb'))

# Loading model to compare the results
model = pickle.load(open('model.pkl','rb'))
print(model.predict([[2, 9, 6]]))
```

MACHINE LEARNING



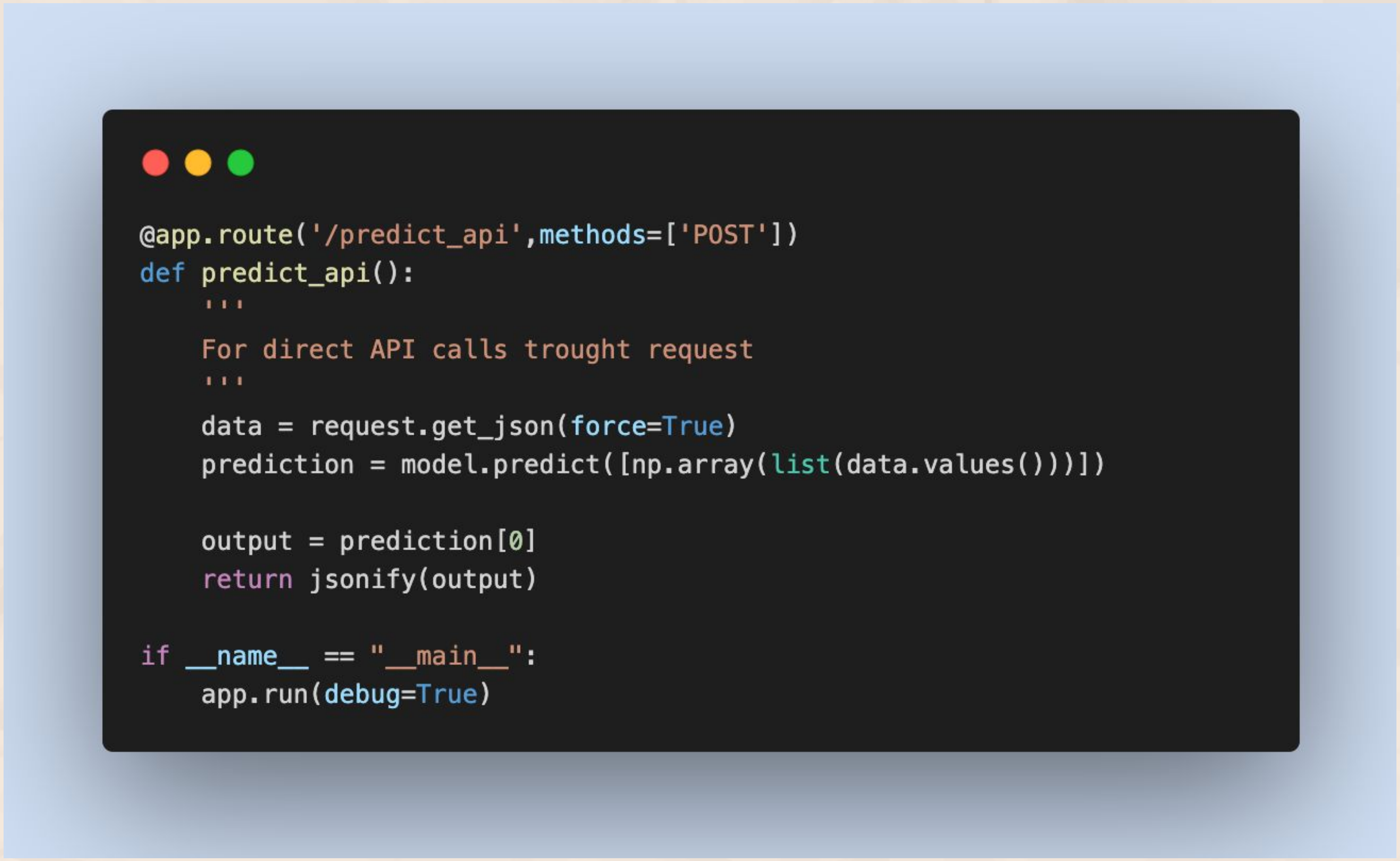
```python
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle
from flask import Flask, request, jsonify
from flask_restx import Api, Resource, fields
from werkzeug.middleware.proxy_fix import ProxyFix


app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))
app.wsgi_app = ProxyFix(app.wsgi_app)


@app.route('/')
def home():
    return render_template('index.html')
```

MACHINE LEARNING



```python
@app.route('/predict',methods=['POST'])
def predict():
    '''
    For rendering results on HTML GUI
    '''
    int_features = [int(x) for x in request.form.values()]
    final_features = [np.array(int_features)]
    prediction = model.predict(final_features)

    output = round(prediction[0], 2)

    return render_template('index.html', prediction_text=
'Employee Salary should be $ {}'.format(output))
```

MACHINE LEARNING



```
@app.route('/predict_api',methods=['POST'])
def predict_api():
    '''
    For direct API calls trought request
    '''
    data = request.get_json(force=True)
    prediction = model.predict([np.array(list(data.values()))])

    output = prediction[0]
    return jsonify(output)

if __name__ == "__main__":
    app.run(debug=True)
```

Thank you