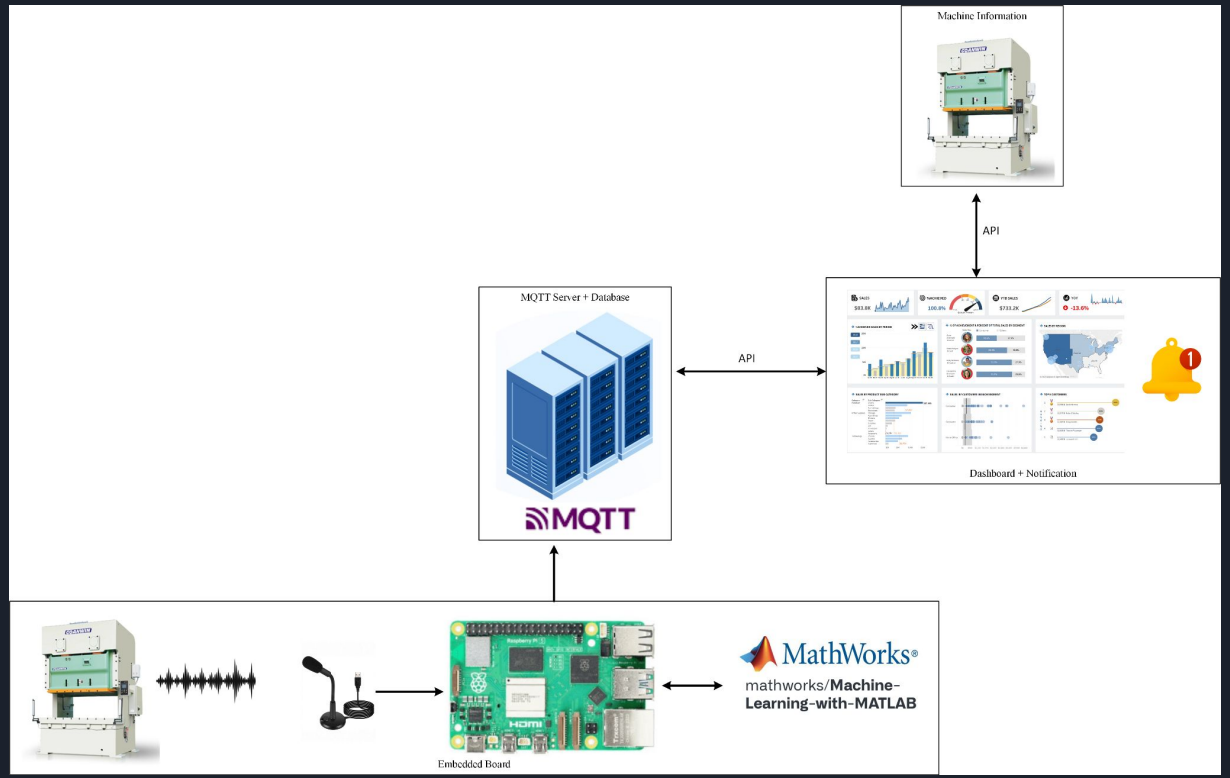
Server Programming

Korawit Orkphol, D.Eng
Department of Computer Engineering
Faculty of Engineering at Sriracha
Kasetsart University Sriracha Campus

System Overview



System Components

- MQTT
- API & Services
- Database
- Dashboard

MQTT Broker (1)

- Create a folder and subfolder
  /mosquitto/config
  /mosquitto/data
  /mosquitto/log
- Save the config file mosquitto.conf to /mosquitto/config

```
listener 1883 0.0.0.0
allow_anonymous true
persistence true
persistence_location /mosquitto/data/
log_dest file /mosquitto/log/mosquitto.log
log_dest stdout
```

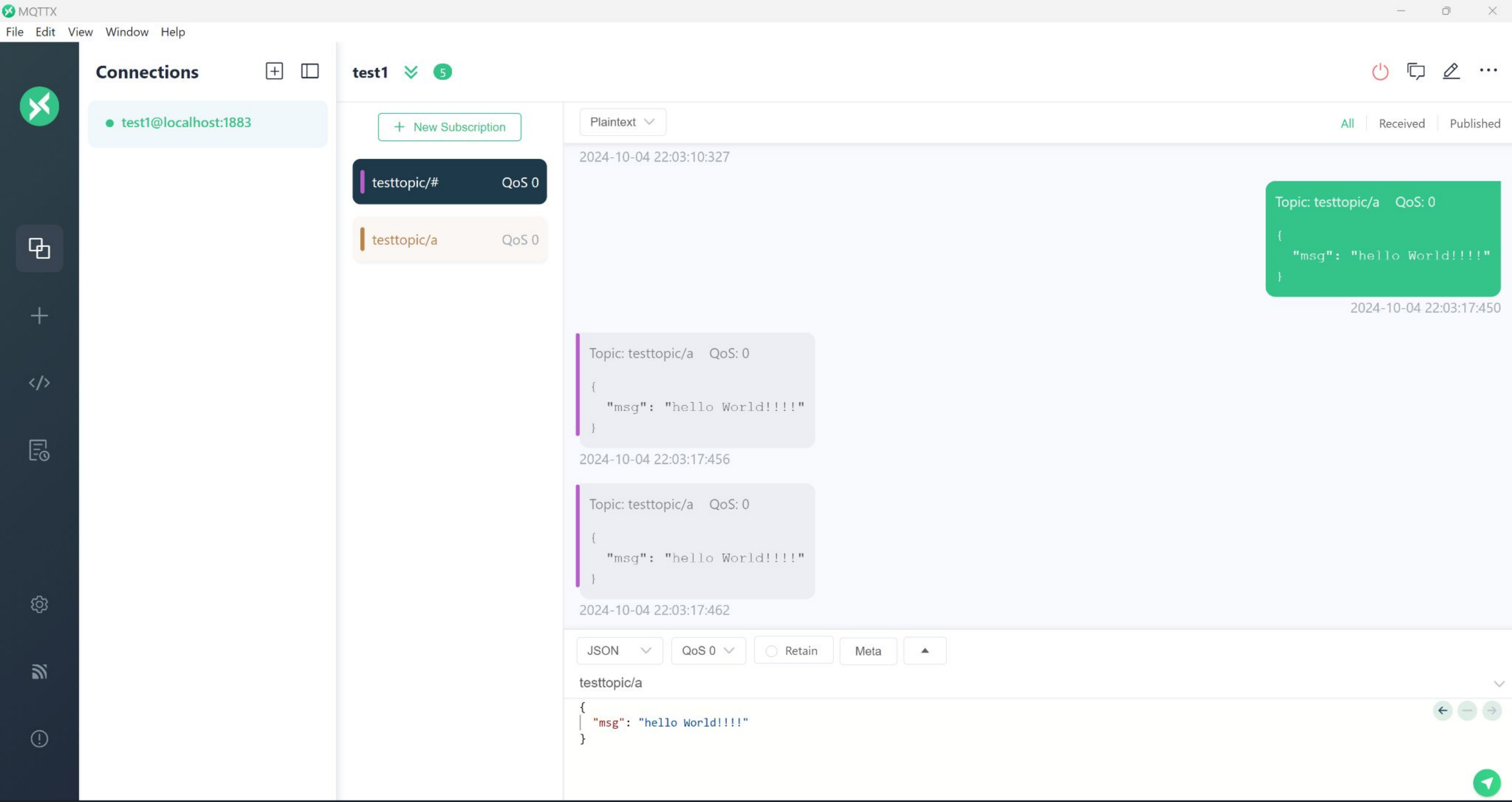
MQTT Broker (2)

- Create a container from image:
  $docker run -dp 1883:1883 -v ./mosquitto:/mosquitto/ --restart always eclipse-mosquitto
- Test MQTT Broker using MQTTX
  - subscribe topics
  - publish message

Yes, you can run VirtualBox along with Docker Desktop if you have enabled the Windows Hypervisor Platform feature on your machine. Why is **Windows 10 or Windows 11** required? Docker Desktop uses the Windows Hyper-V features. While older Windows versions have Hyper-V, their Hyper-V implementations lack features critical for Docker Desktop to work.

Important

To run Windows containers, you need Windows 10 or Windows 11 Professional or Enterprise edition. Windows Home or Education editions only allow you to run Linux containers.


MQTTX
File Edit View Window Help
Connections
test1@localhost:1883
test1
5
New Subscription
testtopic/# QoS 0
testtopic/a QoS 0
Plaintext
All
Received
Published
2024-10-04 22:03:10:327
Topic: testtopic/a QoS: 0
{
"msg": "hello World!!!!"
}
2024-10-04 22:03:17:450
Topic: testtopic/a QoS: 0
{
"msg": "hello World!!!!"
}
2024-10-04 22:03:17:456
Topic: testtopic/a QoS: 0
{
"msg": "hello World!!!!"
}
2024-10-04 22:03:17:462
JSON
QoS 0
Retain
Meta
testtopic/a
{
"msg": "hello World!!!!"
}

Example for publishing messages

```
import paho.mqtt.client as mqtt
import time

broker_hostname = "localhost"
port = 1883

def on_connect(client, userdata, flags, return_code):
    if return_code == 0:
        print("connected")
    else:
        print("could not connect, return code:", return_code)

client = mqtt.Client("Client1")
# client.username_pw_set(username="user_name", password="password") # uncomment if you
client.on_connect = on_connect

client.connect(broker_hostname, port)
client.loop_start()

topic = "Test"
msg_count = 0
```

Example for publishing messages

```
try:
    while msg_count < 10:
        time.sleep(1)
        msg_count += 1
        result = client.publish(topic, msg_count)
        status = result[0]
        if status == 0:
            print("Message "+ str(msg_count) + " is published to topic " + topic)
        else:
            print("Failed to send message to topic " + topic)
            if not client.is_connected():
                print("Client not connected, exiting...")
                break
finally:
    client.disconnect()
    client.loop_stop()
```

Example code for creating a client subscribing to messages

```
import paho.mqtt.client as mqtt
import time

def on_connect(client, userdata, flags, return_code):
    if return_code == 0:
        print("connected")
        client.subscribe("Test")
    else:
        print("could not connect, return code:", return_code)
        client.failed_connect = True


def on_message(client, userdata, message):
    print("Received message: ", str(message.payload.decode("utf-8")))


broker_hostname ="localhost"
port = 1883
```

Example code for creating a client subscribing to messages

```
client = mqtt.Client("Client2")
# client.username_pw_set(username="user_name", password="password") # uncomment if you
client.on_connect = on_connect
client.on_message = on_message
client.failed_connect = False

client.connect(broker_hostname, port)
client.loop_start()

# this try-finally block ensures that whenever we terminate the program earlier by hitt
try:
    i = 0
    while i < 15 and client.failed_connect == False:
        time.sleep(1)
        i = i + 1
    if client.failed_connect == True:
        print('Connection failed, exiting...')

finally:
    client.disconnect()
    client.loop_stop()
```

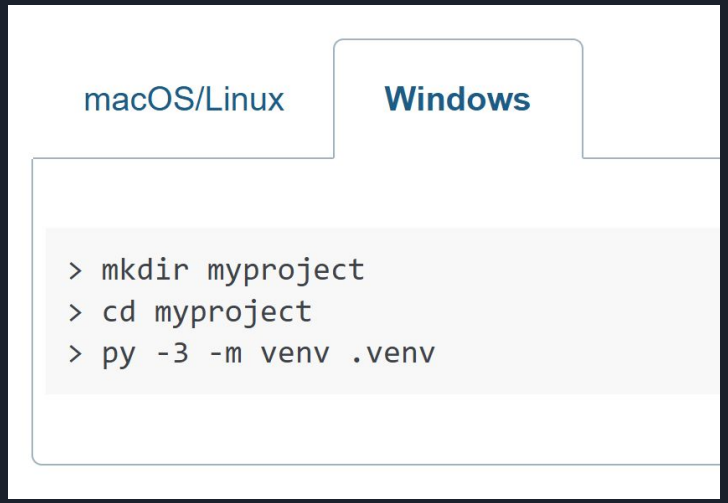
References

hub.docker.com/_/eclipse-mosquitto/

Mosquitto Docker Configuration - Ultimate Guide | Cedalo

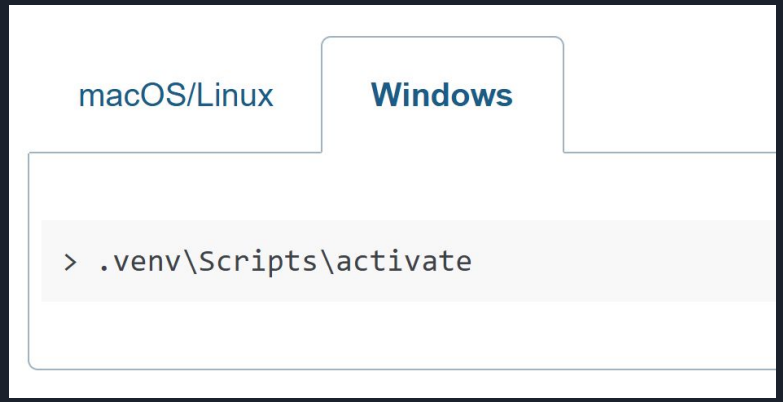
API & Service

- You can write an API using any languages such as Nodejs, flask
- For precamp, I would like to demonstrate the restful API using flask
  - Create virtual environment



API & Service

- Activate the environment



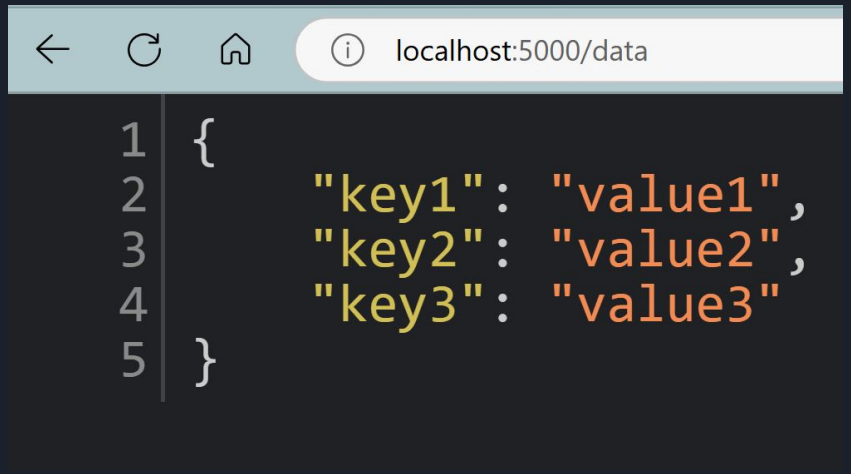
- then install flask

  > pip install Flask

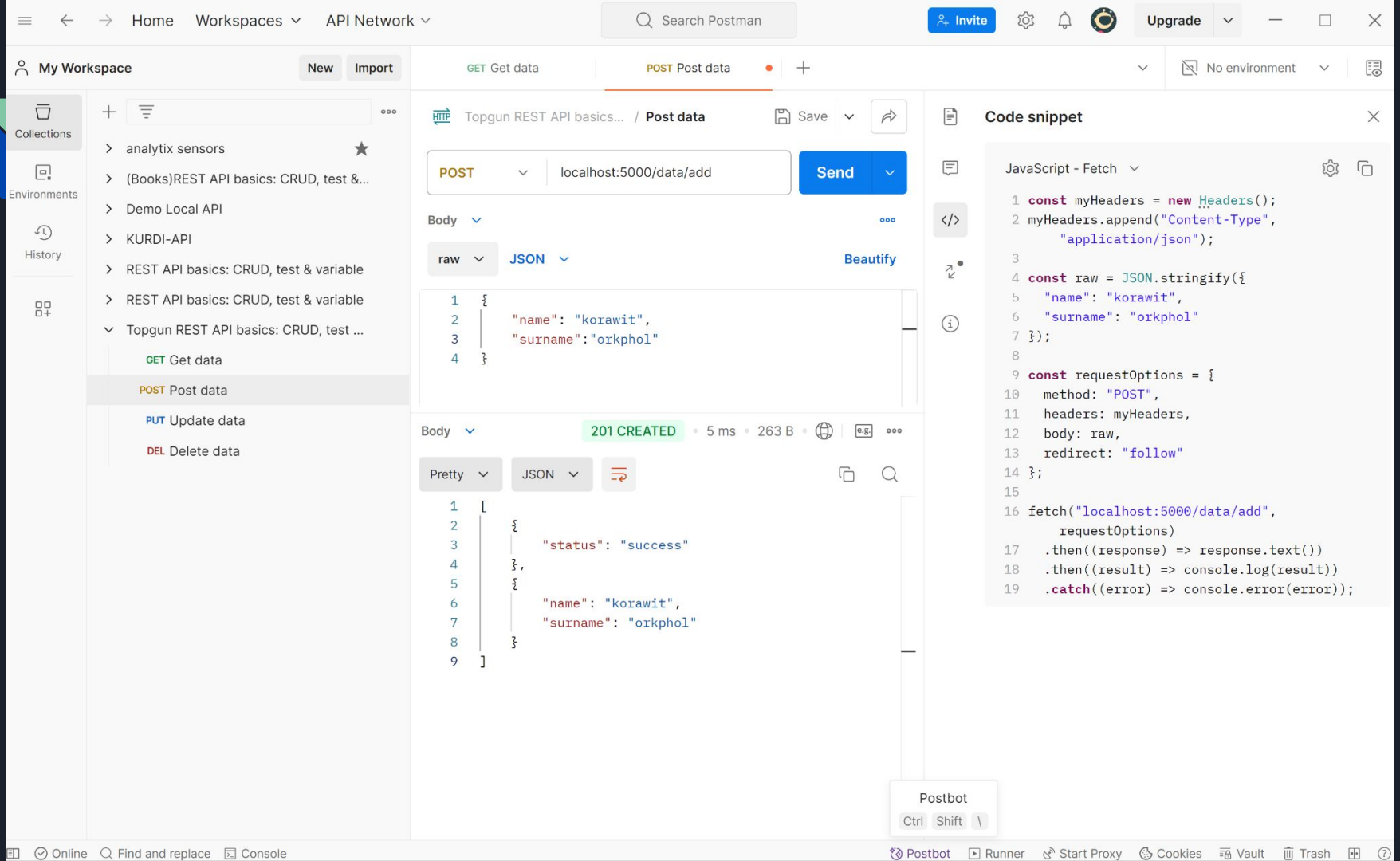
Example code of Flask API

```python
from flask import request,Flask,jsonify,make_response

app = Flask(__name__)

@app.route('/')
def index():
    return "<h1>Hello World</h1>"

@app.route("/data",methods=["GET"])
def get_data():
    data={"key1":"value1","key2":"value2","key3":"value3"}
    return jsonify(data),200

if __name__ == '__main__':
    app.run(host="0.0.0.0",port=5000,debug=True)
```

Testing Flask API



You need to implement other endpoints and methods as you need such as POST PUT PATCH DELETE


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
GET Get data
POST Post data
No environment
Collections
Environments
History
analytix sensors
(Books)REST API basics: CRUD, test &...
Demo Local API
KURDI-API
REST API basics: CRUD, test & variable
REST API basics: CRUD, test & variable
Topgun REST API basics: CRUD, test ...
GET Get data
POST Post data
PUT Update data
DEL Delete data
Topgun REST API basics... / Post data
Save
POST
localhost:5000/data/add
Send
Body
raw
JSON
Beautify
{
"name": "korawit",
"surname":"orkphol"
}
Body
201 CREATED
5 ms
263 B
Pretty
JSON
[
{
"status": "success"
},
{
"name": "korawit",
"surname": "orkphol"
}
]
Code snippet
JavaScript - Fetch
const myHeaders = new Headers();
myHeaders.append("Content-Type",
"application/json");
const raw = JSON.stringify({
"name": "korawit",
"surname": "orkphol"
});
const requestOptions = {
method: "POST",
headers: myHeaders,
body: raw,
redirect: "follow"
};
fetch("localhost:5000/data/add",
requestOptions)
.then((response) => response.text())
.then((result) => console.log(result))
.catch((error) => console.error(error));
Postbot
Ctrl
Shift
\
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

Run API using docker

1. Create Dockerfile

   ```
   FROM python:3

   WORKDIR /app

   COPY requirement.txt .
   COPY api.py .

   RUN pip install --no-cache-dir --upgrade pip && \
   pip install --no-cache-dir -r requirement.txt

   CMD ["python", "api.py"]

   EXPOSE 5000
   ```

2. Create Requirement.txt
   > .\venv\Scripts\activate
   > pip freeze > requirement.txt

Run API using docker

3. Create an image
   > sudo docker build -t api_image .
4. Running a container
   > sudo docker run -dp 0.0.0.0:5000:5000 --name api --restart always api_image

References

Installation — Flask Documentation (2.3.x) (palletsprojects.com)

Database

You can use any databases but for precamp, I would like to demonstrate Postgresql and Mongodb

1. For running postgresql container :
   > docker run -dp 5432:5432 --name postgresql -v pgdata:/var/lib/postgresql/data -e POSTGRES_PASSWORD=secret --restart always postgres:latest
2. Using psql client to connect db
   > psql -h localhost -U postgres
3. Write sql command to work with db
4. Writing any script to manipulate data such as python, javascript, etc.

SQL Commands

- CREATE DATABASE company;
- \c company
- CREATE TABLE employees (

  employee_id serial PRIMARY KEY,

  first_name text NOT NULL,

  last_name text NOT NULL,

  hire_date date

);

- \d employees

SQL Commands : CRUD

- INSERT INTO employees (employee_id, first_name, last_name, hire_date) VALUES (101, 'John', 'Doe', '2024-09-30');
- SELECT * FROM employees WHERE employee_id = 101;
- UPDATE employees SET last_name = 'Smith' WHERE employee_id = 101;
- DELETE FROM employees WHERE employee_id = 101;

Database

1. For running mongodb container:
   > docker run -d --name mongodb -p 27017:27017 -e MONGO_INITDB_ROOT_USERNAME=your_username -e MONGO_INITDB_ROOT_PASSWORD=your_password mongo:latest
2. Testing using mongosh
   >mongosh --host localhost -u your_username
3. Try some command see this cheatsheet
4. If you want to run mongodb on cloud please visit cloud.mongodb.com

Reference : mongo - Official Image | Docker Hub
MongoDB Crash Course 2022 - YouTube

Example code for connecting mongodb (CRUD)

```python
from pymongo.mongo_client import MongoClient

uri = "mongodb+srv://user:pass@cluster0.2owdljr.mongodb.net/?retryWrites=true&w=majority"
#uri ="mongodb://myuser:mypassword@localhost:27017/mydb"

# Create a new client and connect to the server
client = MongoClient(uri)
# Send a ping to confirm a successful connection
try:
    client.admin.command('ping')
    print("Pinged your deployment. You successfully connected to MongoDB!")
    db = client["students"]
    collection = db["std_info"]
    while True:
        print("===MENU===")
        print("1: show all records")
        print("2: insert record")
        print("3: update record")
        print("4: delete record")
        print("5: exit")
        choice = input("Please choose:")
        choice = int(choice)
```

```python
if choice==1:
    print(f'found {collection.count_documents({})} records')
    all_students = collection.find()
    for std in all_students:
        print(std)
elif choice==2:
    id=input("Input student id:")
    name=input("Input fullname:")
    major=input("Input major:")
    gpa=input("Input gpa:")
    gpa=float(gpa)
    try:
        collection.insert_one({"_id":id,
                               "fullname":name,
                               "major":major,
                               "gpa":gpa
                               })
    except Exception as e:
        print(e)
```

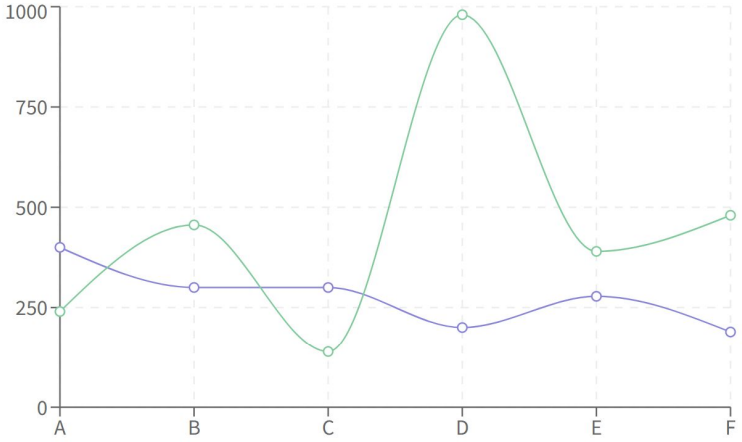
```
        elif choice==3:
            student_id_to_update = input("Input the student ID to update: ")
            new_name=input("Input new_fullname:")
            new_major=input("Input new_major:")
            new_gpa=input("Input new_gpa:")
            new_gpa=float(new_gpa)
            try:
                collection.update_one(
                    {"_id": student_id_to_update},
                    {"$set": {
                        "fullname": new_name,
                        "major": new_major,
                        "gpa": new_gpa
                    }})
            except Exception as e:
                print(e)
        elif choice==4:
            student_id_to_delete = input("Input the student ID to delete: ")
            try:
                collection.delete_one(
                    {"_id": student_id_to_delete})
            except Exception as e:
                print(e)
        elif choice==5:
            break
except Exception as e:
    print(e)
finally:
    client.close()
```

Dashboard

You can implement Dashboard using any web programming languages, but for pre-camp, I would like to demonstrate using React

- install nodejs
- and run
  >npx create-react-app dashboard
  >cd dashboard
  >npm start
- or using on github codespace

Rechart.js




```
<LineChart width={500} height={300} data={data}>
  <XAxis dataKey="name"/>
  <YAxis/>
  <CartesianGrid stroke="#eee" strokeDasharray="5 5"/>
  <Line type="monotone" dataKey="uv" stroke="#8884d8" />
  <Line type="monotone" dataKey="pv" stroke="#82ca9d" />
</LineChart>
```

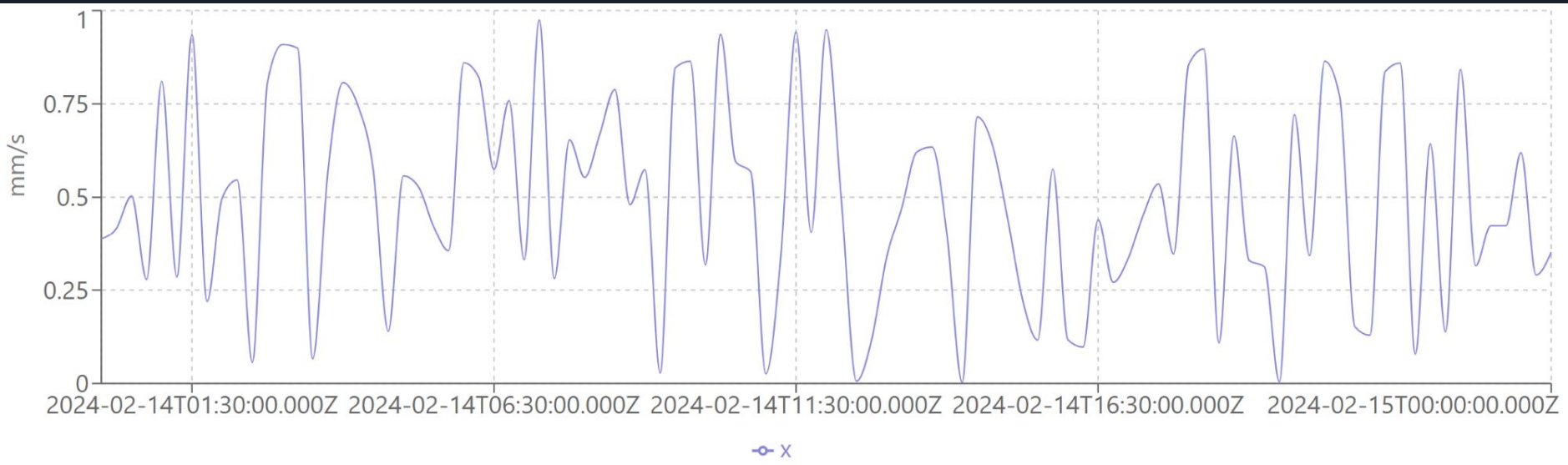
$npm install recharts

App.jsx

src > App.jsx > App

```jsx
import './App.css';
import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, Legend, Label } from 'recharts';

function generateRandomData() {
  const startDate = new Date('2024-02-14T00:00:00Z'); // Start date
  const endDate = new Date('2024-02-15T00:00:00Z'); // End date (two days later)
  const intervalMinutes = 15; // Time interval in minutes

  const data = [];
  let currentTime = startDate;

  while (currentTime <= endDate) {
      const timestamp = currentTime.toISOString(); // Convert to ISO string
      const x = Math.random() // Random value for x

      data.push({ timestamp, x });

      // Increment time by the specified interval
      currentTime = new Date(currentTime.getTime() + intervalMinutes * 60 * 1000);
  }

  return data;
}
```

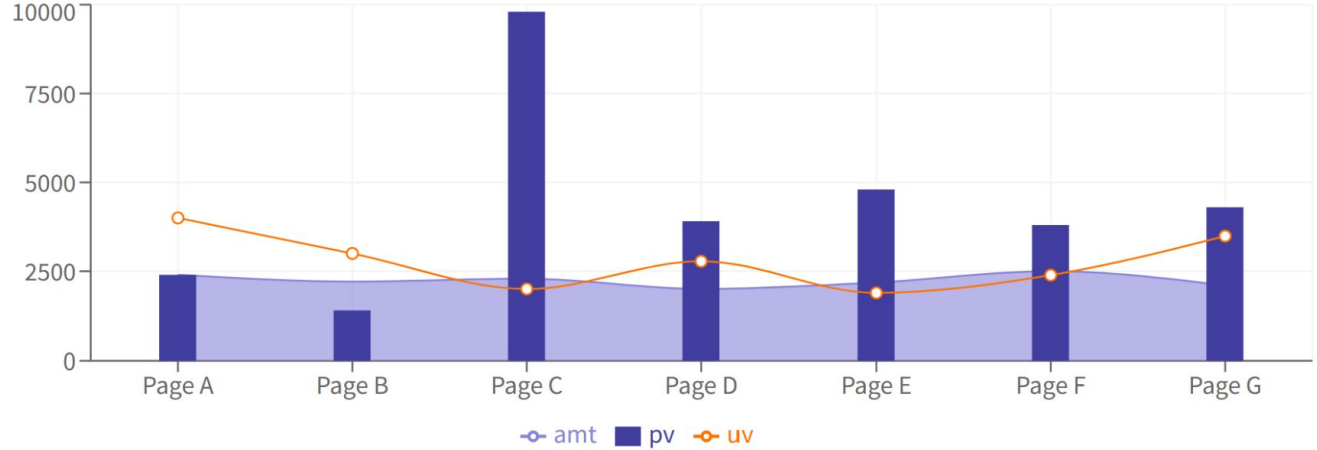
```
function App() {

  const data = generateRandomData();
  return (<>
    <LineChart width={1000} height={300} data={data}>
    <CartesianGrid strokeDasharray="3 3" />
    <XAxis dataKey="timestamp" />
    <YAxis yAxisId="left">
        <Label value="mm/s" position="insideLeft" angle={-90} />
    </YAxis>
    <Tooltip />
    <Legend />
    <Line type="monotone" dataKey="x" stroke="#8884d8" yAxisId="left" dot={false} />
    </LineChart></>
  );
}

export default App;
```



Example of Rechart Graph Rendering

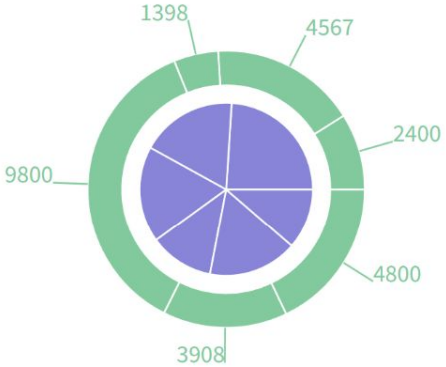
ComposedChart

A chart composed of line, area, and bar charts. When you just want to draw a chart of a single type like line, then LineChart is recommended.




```
<ComposedChart width={730} height={250} data={data}>
  <XAxis dataKey="name" />
  <YAxis />
  <Tooltip />
  <Legend />
  <CartesianGrid stroke="#f5f5f5" />
  <Area type="monotone" dataKey="amt" fill="#8884d8" stroke="#8884d8" />
  <Bar dataKey="pv" barSize={20} fill="#413ea0" />
  <Line type="monotone" dataKey="uv" stroke="#ff7300" />
</ComposedChart>
```

PieChart




```
<PieChart width={730} height={250}>
  <Pie data={data01} dataKey="value" nameKey="name" cx="50%" cy="50%" outerRadius={50} fill="#8884d8" />
  <Pie data={data02} dataKey="value" nameKey="name" cx="50%" cy="50%" innerRadius={60} outerRadius={80} fill="#82ca9d" label />
</PieChart>
```

More on Rechart API…

Realtime Notification : Socket.io

Socket.IO is an event-driven library for real-time web applications. It enables real-time, bi-directional communication between web clients and servers. It consists of two components: a client, and a server. Both components have a nearly identical API.

see

1. Tutorial - Introduction | Socket.IO
2. How to use with React | Socket.IO

```
import React, { useEffect, useState } from 'react';
import io from 'socket.io-client';

const socket = io('http://localhost:5000', {
    autoConnect: true
  }); // Replace with your server URL

function Socket() {
  const [message,setMessage]=useState([]);

  useEffect(() => {
    // no-op if the socket is already connected
    socket.connect();

    return () => {
      socket.disconnect();
    };
  }, []);
```

FRONT-END

```
  useEffect(() => {
    socket.on('response', (response) => {
        console.log('Server response back:',response);
        setMessage([...message,response]);
      });
    return () => {
        socket.off("response");
      };
  }, [message]);

  function sendMessage(){
    socket.emit('message',"Hello server");
  }
  const messageList=message.map((_message,_index)=><li>{_index} {_message}</li>);
  return (<div className="App"><ul>{messageList}</ul><button onClick={sendMessage}>Greet</button></div>);
}
export default Socket;
```

```python
from flask import Flask
from flask_socketio import SocketIO, emit
from datetime import datetime
import time
app = Flask(__name__)
socketio = SocketIO(app,cors_allowed_origins="*")

@app.route('/')
def index():
    return "<h1>Hello World</h1>"

@socketio.on('connect')
def handle_connect():
    print('Client connected')
```

BACK-END

```python
@socketio.on('message')
def handle_message(data):
    print('Received message:', data)
    for i in range(10):
        now = datetime.now()
        socketio.emit('response', 'You sent {0} {1}'.format(data,str(now)))
        time.sleep(5)

if __name__ == '__main__':
    socketio.run(app, debug=True)
```

0 You sent Hello server 2024-10-05 00:07:47.590342
1 You sent Hello server 2024-10-05 00:07:52.595233
2 You sent Hello server 2024-10-05 00:07:57.608479
3 You sent Hello server 2024-10-05 00:08:02.609202
4 You sent Hello server 2024-10-05 00:08:07.623158
5 You sent Hello server 2024-10-05 00:08:12.636251
6 You sent Hello server 2024-10-05 00:08:17.640756
7 You sent Hello server 2024-10-05 00:08:22.650027
8 You sent Hello server 2024-10-05 00:08:27.663556
9 You sent Hello server 2024-10-05 00:08:32.668894

Greet

Dockerfile for React front-end

```
FROM node:latest

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build --production

RUN npm install -g serve

EXPOSE 3000

CMD serve -s build
```

Thank you Q&A